

InRule for Microsoft Dynamics 365 Deployment Guide



Document Updated against InRule v5.8.1

Document Updated against InRule for Microsoft Dynamics 365 Integration Framework v2.3.21

Document Updated against Microsoft Dynamics 365 v9.1 online, v9.0 on-prem

Solution Built against Microsoft .NET Framework v4.7.2

InRule does not upgrade this document after each Integration Framework release, please see release notes for individual versions if the version that you are using does not match the versions listed above.

If you are working with earlier versions of any of the above products, the information in this document may not apply to you. Please check to see if earlier documentation is available to cover your needs.

CONFIDENTIAL Any use, copying or disclosure by or to any other person than has downloaded a trial version of InRule or signed an NDA is strictly prohibited. If you have received this document by any other means than a download or an email from an InRule employee, please destroy it retaining no electronic or printed copies.

© Copyright 2024 InRule Technology, Inc.

Microsoft®, Microsoft Dynamics® and the Microsoft Dynamics Logo are registered trademarks of Microsoft Corporation.

All rights reserved. No parts of this work may be reproduced in any form or by any means – graphic, electronic, or mechanical, including photocopying, recording, taping, or information storage and retrieval systems – without the written permission from InRule Technologies, Inc.

InRule, InRule Technology, irAuthor, irVerify, irServer, irCatalog, irSDK and irX are registered trademarks of InRule Technology, Inc. All other trademarks and trade names mentioned herein may be the trademarks of their respective owners and are hereby acknowledged.

Table of Contents

Table of Contents	2
1 Introducing InRule® for Microsoft Dynamics 365	4
2 Understanding your options	5
2.1 Microsoft Dynamics 365 (Online) – InRule SaaS	5
2.2 Microsoft Dynamics 365 (Online) – Self-hosted in Microsoft Azure	6
2.3 Microsoft Dynamics 365 (On Premises) – Local IIS or Azure IaaS	6
2.4 Other Integration Scenarios	7
3 Performing the Installation: In Azure	8
3.1 An overview of the Components needed	8
3.2 Gathering prerequisites	9
3.3 Deploying and Configuring Components	14
3.3.1 Catalog App Service	14
3.3.2 Rule Execution App Service for Dynamics 365	17
3.3.3 InRule Solution for Dynamics 365	25
4 Performing the Installation: On-Premises	35
4.1 An overview of the Components needed	35
4.2 Gathering prerequisites	36
4.3 Deploying and Configuring Components	38
4.3.1 Setting Up an Application Pool in IIS	38
4.3.2 Setting Up the IIS Site for the Rule Execution Web Service	40
4.3.3 Deploy the Rule Execution Web Service	43
4.3.4 InRule Solution for Dynamics On-Prem Deployment	45
Appendix A: Additional Resources	50
InRule’s Support Website	50
InRule’s Support Team	50
InRule’s ROAD Team	50
Appendix B: Anatomy of a Request for Execution of Rules Diagram	51
Appendix C: General Authoring and Integration Concepts	52
Appendix D: Accessing Dynamics 365 Directly from Rule Helper	53
Appendix E: Methods for Executing Rules from Dynamics 365 and Power Platform	62
1 Dataverse Events	65
2 Rules Engine Action	67
3 Run Rules Button	68

4	Workflow Activity	70
5	Form Events	71
6	JavaScript	75
7	Power Platform.....	76
Appendix F: Rules Configuration and Settings		86
Appendix G: Endpoint Override Configuration.....		99
Appendix H: Azure App Service Plan & Application Insights Configuration		102
Appendix I: InRule SaaS Portal Configuration		105
Appendix J: Dynamics 365 Tracing and InRule Event Logging.....		108
Appendix K: License Management		117
Appendix L: Redeploying and Upgrading Versions		122
Appendix M: Known Issues, Limitations and Troubleshooting		126
Configuration		126
Execution Runtime		126
Roles, Privileges and Access.....		127
Usability		129
Performance		130
Request and Response Limitations		137
Deployment and Setup.....		138
Miscellaneous Troubleshooting Items.....		139

1 Introducing InRule® for Microsoft Dynamics 365

InRule for Microsoft Dynamics 365 enables rich rule integration with Microsoft Dynamics 365 and Microsoft Power Platform. The end-to-end solution is comprised of:

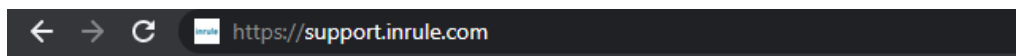
- **irAuthor® with irX for Dynamics 365** – extension to irAuthor, InRule's premium desktop rule authoring tool
- **InRule Solution for Dynamics 365** – available from [Microsoft AppSource](#)
- **Rule Execution Services for Dynamics 365** - InRule SaaS, self-hosted in Azure, or other

This guide focuses on the deployment of the Rule Execution Services for Dynamics and corresponding InRule Dynamics Solution to your environment. The InRule Dynamics Solution may be deployed directly from the Microsoft AppSource (recommended in most cases) or from the Integration Framework zip file downloaded from the [InRule Support Site](#). This guide details the primary deployment paths for cloud-based integration in Microsoft Azure® as well as on-premises Dynamics.

Before beginning this guide, you may first want to familiarize yourself with the *irX for Microsoft Dynamics 365* product by reading the [irX for Microsoft Dynamics 365 Help Documentation](#). This irAuthor extension will allow you to author rules against Dynamics entities and become familiar with the types of rules-driven processes that can be implemented. After testing locally from your desktop using irVerify with either your own rule app or the [DynamicsRules sample rule app](#), the rules will be ready for execution from Dynamics or Power Platform. At this point, this guide will become highly relevant for deploying the InRule solution and services to establish the selected integration patterns.

There are several options available when it comes to choosing how to integrate InRule with Dynamics 365 or Power Platform. Beyond Dynamics, virtually any Power App (Model-driven & Canvas) or Power Automate Flow that leverages the Dataverse can benefit from event-driven or on-demand rule execution. With InRule and Power Platform, business users have unprecedented flexibility to automate end-to-end business processes such as claims processing, product eligibility, CPQ pricing models and countless other scenarios. This document provides an addendum, [Appendix E: Methods for Executing Rules from Dynamics 365 and Power Platform](#) that discusses the different options available for running rules beyond what the primary deployment steps cover. It is a good next step to review for implementers who are looking for advanced options to address their specific decision automation concerns.

Additional material is available on the Downloads section of our support website. Please see the [Additional Resources](#) section of this document for support website detail.



InRule® for Microsoft Dynamics

Enterprise-grade rules for Microsoft Dynamics 365

[irX® for Microsoft Dynamics 365 Help Documentation \(1.2 MB\)](#)

[InRule® for Microsoft Dynamics 365 Deployment Guide \(5 MB\)](#)

[InRule® for Microsoft Dynamics 365 Integration Framework \(22.7 MB\)](#)

2 Understanding your options

This guide is oriented towards deploying InRule as a cloud-based solution in Microsoft Azure to interoperate with Microsoft Dynamics 365 Online. Alternatively, Dynamics 365 on-premises version 9.0 and up is still supported via local installation of the InRule services.

The following sections describe which deployment scenario will be the most applicable depending on your needs:

- [2.1 Microsoft Dynamics 365 \(Online\) – InRule SaaS-hosted Rule Execution Service](#)
- [2.2 Microsoft Dynamics 365 \(Online\) – Self-hosted Rule Execution Service in Microsoft Azure](#)
- [2.3 Microsoft Dynamics 365 \(On Premises\) – Local IIS or Azure IaaS Rule Execution Service](#)
- [2.4 Other Integration Scenarios](#)

The primary consideration for InRule SaaS vs self-hosted Azure generally depends on if your organization is already setup to manage an Azure subscription and services.

- **If your organization does not have an Azure subscription**, this can be compelling rationale to go with InRule SaaS and forgo the overhead of Azure management and deployment.
- **If your organization has an Azure subscription**, it does not exclude you from going with InRule SaaS, but it may indicate that your IT department has requirements or preferences for self-managed Azure solutions.

Either way, this guide will provide you with the information to help determine which InRule configuration will best suit your needs, including Government Cloud and other considerations.

2.1 Microsoft Dynamics 365 (Online) – InRule SaaS

InRule for Dynamics 365 is now available via [InRule SaaS](#), in which case many of the deployment steps in this document are not applicable and will be managed for you. This is the most stream-lined deployment available to both qualified Trial users and licensed customers.



Important: If you are new to InRule and do not have an InRule SaaS subscription, you can request a [free trial here](#) and specify that you would like to integrate it with your Dynamics 365 instance.

With InRule SaaS, the deployment steps are significantly reduced by eliminating the need to install the InRule App Services in Azure. This narrows the deployment focus down to irAuthor with irX for Dynamics 365 and the InRule Dynamics Solution from the Microsoft AppSource. The main deployment steps in this scenario are summarized as follows:

Deployment Step	Installation Type
1. irAuthor with irX for Dynamics 365	InRule installer via the InRule Support Site
2. Rule Execution Services for Dynamics 365	*managed by InRule SaaS Support
3. InRule Solution for Dynamics 365	Microsoft AppSource (or script-based)

All information to set up the connectivity between Rule Execution Services hosted by InRule SaaS and your organization's Dynamics instance can be managed through the InRule SaaS Portal. The following configuration will be managed through the Portal. More information can be found in [Appendix I: InRule SaaS Portal Configuration](#).

Entered by users in the SaaS Portal – configuration information for the Rule Execution App Services

- The Organization URL for Dynamics.
- Service account credentials for Dynamics – used to allow the InRule SaaS-hosted Rule Execution Service to connect to Dynamics. Alternatively, InRule can be granted consent as a Trusted App to your Azure AD (InRule Support to provide specific instructions).

Provided to users in the SaaS Portal – configuration information for the InRule Dynamics Solution:

- SAS Key - Azure Relay key that the Rule Execution App Service is configured to listen on
- Azure SB Namespace

The net effect of an InRule SaaS deployment is that after the above pre-requisites are met, you get to **skip ahead** to Section 3 [Configuring the InRule Solution for Dynamics 365](#). Beyond this initial setup the remainder of the guide will be beneficial for detailing [advanced rule execution methods](#) and [troubleshooting](#).

2.2 Microsoft Dynamics 365 (Online) – Self-hosted in Microsoft Azure

If you are using Microsoft Dynamics 365 (Online), you will be hosting InRule using a Platform-As-A-Service (PaaS) model on Microsoft Azure. When setting up the InRule App Services, ensure that all Azure components and the Dynamics installation are in the same geographical location to reduce timeout exceptions due to network latency. The Azure resources can be deployed using either the ARM Template shipped in the integration zip or via the Azure Marketplace. You can find more info about these options in [Section 3.3](#)

This document discusses deploying the following four components:

1. Catalog App Service and Azure SQL Database
2. Azure Relay
3. Rule Execution App Service for Dynamics 365
4. InRule Solution for Dynamics 365

Section 3 of this document, [Performing the Installation: In Azure](#), provides a complete walkthrough.

2.3 Microsoft Dynamics 365 (On Premises) – Local IIS or Azure IaaS

If you are using Microsoft Dynamics 365 (On premises) or plan to deploy to the cloud using an Infrastructure-As-A-Service (IaaS) model, you will be hosting our integration framework via Internet Information Services (IIS) on a Window Server OS. This document discusses the following 3 components:

1. Catalog Web Service and Database
2. Rule Execution Web Service for Dynamics 365 On-Premises
3. InRule Solution for Dynamics 365 On-Premises

Section 4 of this document, [Performing the Installation: On-Premises](#), provides a complete walkthrough.

2.4 Other Integration Scenarios

You are welcome to look at alternate integration options. The following are some possible integration scenarios that you may pursue, which may require either deployment or solution customization. Please take the opportunity to verify the rationale for customization with InRule to ensure plausibility.

- RuleHelper only integration with the InRule native (non-Dynamics) rule execution service
- Limited Integration with Microsoft Dynamics 365 (Online) without the use of an Azure Relay

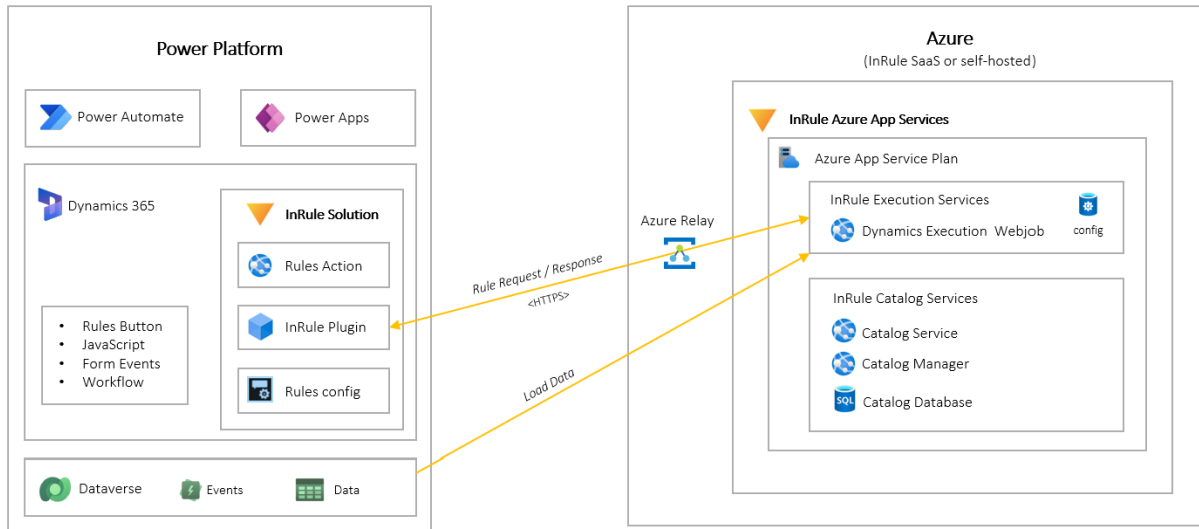
While all of these are possible routes, they will not work with the Integration Framework we provide off the shelf. As such, we strongly encourage that you follow the model outlined in this guide to start with, until you've acquired a more advanced understanding of how all of the pieces of the *InRule® for Microsoft Dynamics 365 Integration Framework* work together to provide a solution.

3 Performing the Installation: In Azure

This section discusses the steps needed to integrate InRule with Microsoft's online version of Microsoft Dynamics 365. Using this scenario InRule components are hosted on Microsoft Azure.

3.1 An overview of the Components needed

This guide will provide the instructions for setting up all of the components below:



Rule Execution App Service for Dynamics 365

The [Rule Execution App Service](#) is responsible for loading Dynamics entity data, executing rules against loaded data, and responding to Dynamics with rule execution results.

InRule Solution for Dynamics 365

The [InRule Solution](#) contains a custom plugin, an endpoint, client resources, a configuration form, and a security role called InRule Integration Administrator. This role is created with read access to all entities for use as a service account, but a different role can be used if desired. The solution must be configured to communicate with the Azure Relay.

Catalog App Service and Azure SQL Database

A Catalog service will be used to store Rule Apps that will be consumed by the Rule Execution App Service. This Catalog Service will be hosted as an Azure App Service. The back end of the Catalog Service utilizes an Azure SQL Database for retrieval and persistence of Rule Applications.

Azure Relay

Dynamics 365 is designed to communicate to third party services through an Azure Relay. Utilizing an Azure Relay is the preferred communication mechanism allowing for security and quick horizontal scaling of services. The *Rule Execution App Service* connects to the Relay and registers itself as a Relay listener. When Dynamics 365 makes a request to the Relay, the Relay relays that message to a listener and allows for two-way communications for as long as the connection is open.

An Azure Relay is a specific kind of Azure Service Bus, that exists as a separate resource type in Azure. Within the context of the InRule architecture, the Relay performs the same function as a “traditional” Service Bus but includes some extra functionality specific to WCF relays. Relays also feature Hybrid Connections, but the *Rule Execution App Service* currently only supports WCF relays. For the sake of consistency with Microsoft’s Dynamics documentation, we will refer to it as a Relay throughout this document. Microsoft’s information about Azure Relay network security can be found [here](#).

3.2 Gathering prerequisites

This section reviews what you will want to have prepared before you begin with the integration steps in the next section.

.NET Framework

The InRule for Microsoft Dynamics 365 integration framework is built against Microsoft .NET Framework v.4.7.2. For on-prem installations of the framework, ensure you have this .NET version installed.

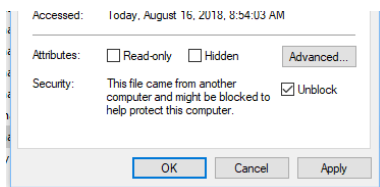
Optional Resource Files

The following file can be downloaded from [our support website’s downloads section](#):

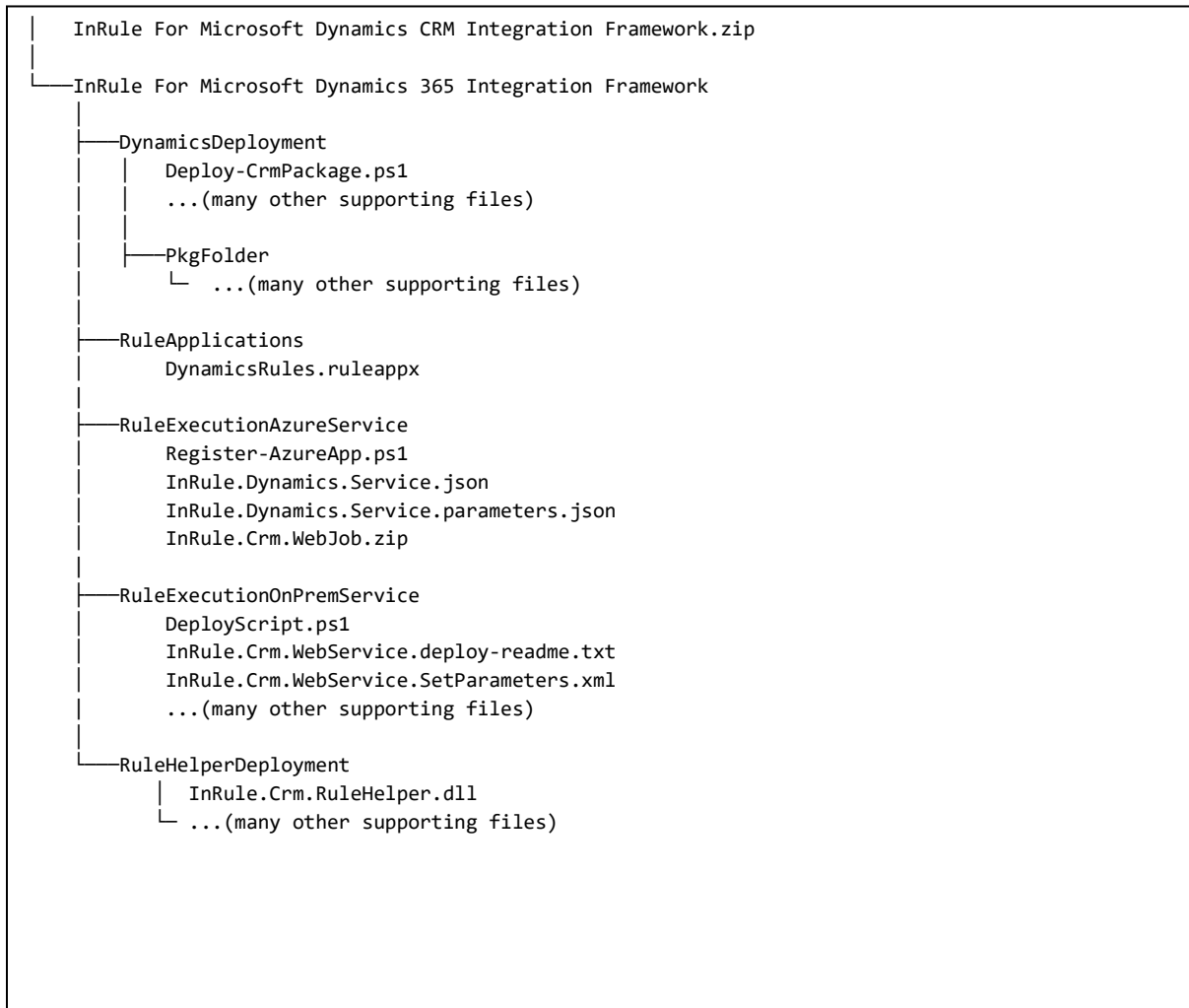
- InRule for Microsoft Dynamics 365 Integration Framework.zip

This zip file contains several resources that can be used in advanced deployments. A typical installation does not require the use of these resources, as all of the required deployment assets are available online. However, certain deployment scenarios do require the assets contained in this zip file – such as: [InRule Solution for Dynamics 365](#).

After you have downloaded this file, but before extracting, make sure that you go to the file properties for the zip and select **Unblock**. If the zip file is not unblocked before extracting, the deployment scripts will not be able to execute successfully.



After unblocking the zip file, extract the contents to a working folder. When you are finished, you should have a directory structure that looks like this:



Rule Authoring Environment

A Rule Authoring Environment is used to upload a Rule Application to your catalog app service. A Rule Authoring Environment is a machine or virtual machine where irAuthor has been installed with the irX for Microsoft Dynamics 365 extension. If you followed the instructions outlined in [irX for Microsoft Dynamics 365 Help Documentation](#) then you should already have a rule-authoring environment available to you.

We have made it a point to call out the rule authoring environment separately because it is important to be aware of the licensing implications of this step. You will need to utilize an irAuthor license and an irX for Microsoft Dynamics 365 license for the duration of this process. If you're a system administrator who does not intend to perform rule authoring duties after the deployment is up and running, you can either choose to borrow an environment from someone who will use a rule authoring environment, or you will want to be sure to deactivate your license when you're finished with your deployment responsibilities.

Administrative Accounts

Dynamics 365 Organization Service URI: You will want to have the root URL of the organization web service exposed by your Dynamics 365 instance. The server URL is usually in the format of "<https://organization-name.crm.dynamics.com>"

Dynamics Service Account Login and Password: You will want to have a username and account created specifically for use by the InRule for Microsoft Dynamics 365 Framework App Service. This account will need a role that has read access to all entities that will be used in running rules. The solution includes the 'InRule Integration Administrator' role, which is set up with read privileges for all base and custom entities when the solution is installed. If this or any role other than 'System Administrator' is used, it is important to note that its permissions are not dynamically updated and will have to be done manually. This user account will not be needed if you decide to create an S2S user account outlines in [Section 3.2: Registering an Azure Active Directory Application](#) of this document.



Important: A Service Account with MFA enabled will not allow the *Rule Execution App Service* to authenticate with Dynamics 365. An S2S authentication will need to be used instead.

Administrative Password to use for SQL Server: You should decide what username and password you want to use for administrative privileges on the SQL Server. You will use this password when following the referenced catalog setup guide.

*** This walkthrough will utilize the above administrative login and password for the Catalog Service to connect to the SQL Server Database. In a more secure environment, a separate SQL User should be created that only has access to the single database needed by the catalog. It is up to the reader of this document to go this more secure route.*

Administrative Password to use for Catalog Service: You should decide what username and password you want to use for administrative privileges within irCatalog. You will use this password when following the referenced catalog setup guide and will need to provide it when deploying the Execution Service.

*** This walkthrough utilizes the default login of 'admin' and password of 'password'. It will be up to the reader to go through the process of utilizing the Catalog Manager to change these credentials to be more secure.*

Administrative Login and Password for Microsoft Azure: You must have a username and password that will be used to perform administrative tasks within Microsoft Azure.

InRule Azure License File

You will need a special .xml file used for licensing InRule in an Azure cloud environment. This may have been provided with your InRule Welcome package. Refer to [Appendix K: License Management](#) for more information. You can contact support@InRule.com if you have additional questions.

Deciding resource names

The following worksheet can be used to decide what to name Azure resources as you go through this Guide.

Many of these resources must have names that are unique in the world; they are hosted on Microsoft Azure and are given domain names that match. We recommend creating a "Base" name that does not exceed 14 characters. We recommend encoding an organization name, an application name, and an environment name into this 'Base' name. For Example:

{ApplicationAbbreviation}{OrganizationAbbreviation}{EnvironmentAbbreviation}

MyAppInRuleDev
12345678901234

You can choose to follow this convention or invent your own.

Resource and Description	Example Name
Base Name	MyAppInRuleDev
Azure Resource Group Name	MyAppInRuleDevResourceGroup
Azure SQL Server Name <i>must be lower case</i>	myappinruledevsqlserver
Catalog Database Name	MyAppInRuleDevCatalogDb
Catalog App Service Name	MyAppInRuleDevCatalogService
Relay Namespace	MyAppInRuleDevRelay
Rule Execution App Service Plan Name	MyAppInRuleDevRuleExecutionAppServicePlan
Rule Execution App Service Name	MyAppInRuleDevRuleExecutionAppService

Registering an Azure Active Directory Application (Optional)

This section is only necessary if you are self-hosting and intend to leverage a server-to-server (S2S) connection between the rule execution service and your Dynamics environment. You can also follow the directions here if you are using OAuth authentication, and do not want to use the Microsoft default app and want to create your own instead.

Server-to-server authentication uses a Dynamics Application User associated to an Azure AD Application for authentication instead of a named user account. This approach can be beneficial as it allows Dynamics integration with InRule without having to purchase an additional full user. Additionally, this will save you from needing to store Dynamics user passwords in Azure. For more information about server-to-server authentication, please refer to the following link: <https://docs.microsoft.com/en-us/dynamics365/customer-engagement/developer/build-web-applications-server-server-s2s-authentication>

Be sure to reference [Appendix M: Known Issues, Limitations and Troubleshooting](#) of this document for considerations regarding the user account and permissions created with this approach.

Registering an Azure AD application is fully automated in the Register-AzureApp.ps1 PowerShell script included in the *RuleExecutionAzureService* folder within the *InRule for Microsoft Dynamics 365 Integration Framework.zip* file downloaded in [Section 3.2: Optional Resource Files](#).

This script requires the Azure AD PowerShell module to be installed on the computer before running the script. If this module is not yet installed, you can install it by opening an admin PowerShell window and executing 'Install-Module AzureAD'.

1: Navigate to the \RuleExecutionAzureService directory:

```
PS C:\> cd .\RuleExecutionAzureService\  
PS C:\RuleExecutionAzureService>
```

2: Execute Register-AzureApp.ps1

The script accepts the parameters “Username” and “Password,” which need to be credentials for an Azure Active Directory Global Administrator. Alternatively, you can run the script without passing in any credentials, and the script will prompt you with an interactive login.

By default, the application is registered in the primary directory the user account belongs to. If the app needs to be registered in a different Azure Active Directory tenant, you can pass in the tenant ID with the optional “TenantId” parameter.

Without a TenantId passed:

```
PS C:\RuleExecutionAzureService> .\Regiser-AzureApp.ps1 -Username admin -Password password
```

With an example TenantId passed:

```
PS C:\RuleExecutionAzureService> .\Regiser-AzureApp.ps1 -Username admin -Password password `
>> -TenantId 29d4c658-27b9-4a52-829d-f2dfe1cc7dfg
```

Observe that no errors occur while the application registration script is running.

When finished, the script will output three values: The application name, application ID, and the secret key. Save these 3 values, particularly the secret key, as there will be no way to retrieve this later. The application ID will be needed when deploying the Rule Execution package, and both the Application ID and Key will be needed when deploying the Rule Execution App Service.

```
Application Details for the InRuleDynamicsCrmIntegration application:
=====
Application Name:      InRuleDynamicsCrmIntegration
Application Id:         [REDACTED]
Secret Key:             [REDACTED]
```

3.3 Deploying and Configuring Components

There are two primary paths for deploying the required InRule components:

1. **Microsoft App Stores:** [Azure Marketplace](#) and [Microsoft AppSource](#) - *(Recommended)* Provides a straightforward, UI-driven process that simplifies deployment and eliminates the need to individually deploy both the Catalog App Service and the Rule Execution App Service. Both the Azure Marketplace (for InRule App Services) and the Microsoft AppSource (for the InRule Dynamics Solution) must be utilized to deploy their respective InRule Apps (see matrix below).
2. **Manual Deployment using ARM Templates and Scripts:** Manually deploy [Azure Resource Management \(ARM\) Templates](#) through either the Azure Portal, Powershell, or Azure CLI; then deploy the InRule Solution via Powershell to your Dynamics 365 instance.

InRule components by deployment path:

InRule Component	Microsoft App Stores	Manual Deployment
Catalog App Service	Azure Marketplace	ARM Template
Rule Execution App Service for Dynamics 365		ARM Template
InRule Solution for Dynamics 365	Microsoft AppSource	Powershell Script

While deploying from Azure Marketplace and AppSource is generally recommended for initial environments; in advanced scenarios the manual deployment resources can be leveraged for multi-environment DevOps automation. Also note, choosing either path does not require you to stay on the same path across all components; as the deployment options can be mixed and matched as situations warrant.

In either deployment path, if you wish to leverage a server-to-server (S2S) connection between the rule execution service and your Dynamics environment, you'll need to first register an Azure Active Directory Application. A walkthrough of this can be found in [Section 3.2: Registering an Azure Active Directory Application](#).

While the following sections are primarily focused on the manual deployment path, there are still relevant config and testing steps which apply to both paths. For Marketplace and AppSource deployments, you may skip the manual deployment steps and go directly to the relevant sub-sections for license management, configuration, and setup verification.

3.3.1 Catalog App Service

This section is only required for those following the ARM Template Deployment Path.

Installing the Catalog App Service

The first major objective for a Dynamics 365 implementation is the deployment of a Catalog service to Microsoft Azure.

During this process, you will be creating:

- An Azure SQL Server
- An Azure SQL Server Database
- An Azure App Service to host the InRule Catalog in the Azure cloud

When you are finished, you should be able to connect to this app service from a locally installed copy of irAuthor, and successfully save a Rule App to the catalog.

The full process of installing the Catalog in Microsoft Azure is outlined in the documentation found on the InRule AzureAppServices GitHub, which can be found here:

<https://github.com/InRule/AzureAppServices/blob/master/README.md#ircatalog-and-ircatalog-manager>

Please ensure you are installing the irCatalog service, not the irServer Rule Execution Service, which is found on the same page and is not compatible with Dynamics 365 integration.

Uploading License File

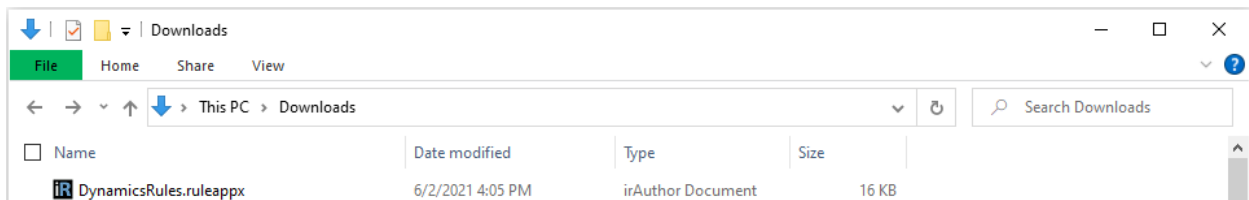
Once you've finished deploying the Catalog Service, you'll need to upload a license file to it in order for the catalog service to function. The simplest way to upload the license file is via the Azure App Service Editor. Alternatively, you can deploy the license file via FTP. A walkthrough of how to upload the license file by either of these methods can be found in [Appendix K: License Management](#).

Testing the Catalog App Service by uploading the sample Rule App

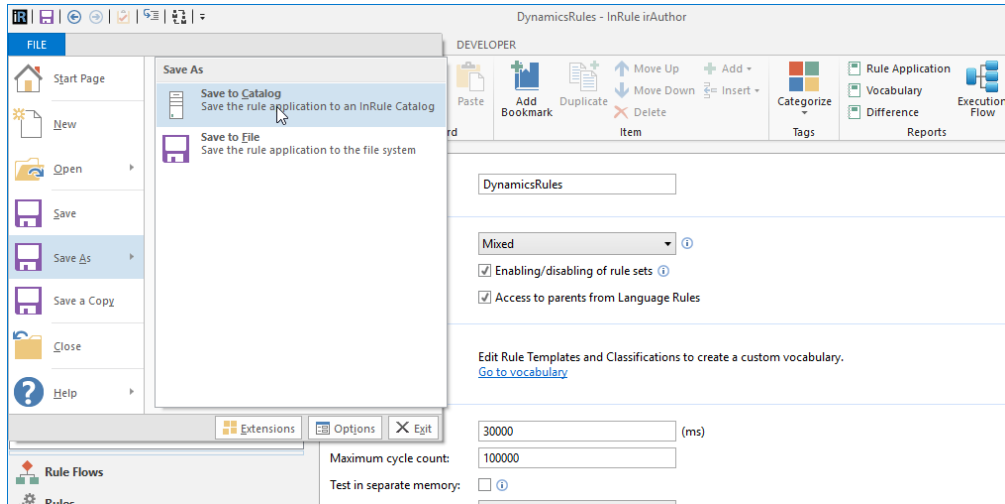
At the conclusion of the installation process outlined above you should have a Catalog URI, Username, and Password to use to connect to the catalog service.

To begin, if you have not already done so, download the [DynamicsRules sample rule app](#). Then utilize irAuthor to upload the rule app to your InRule Catalog. If using InRule SaaS or deploying from the [Azure Marketplace](#), you may already have the sample rule app in your catalog.

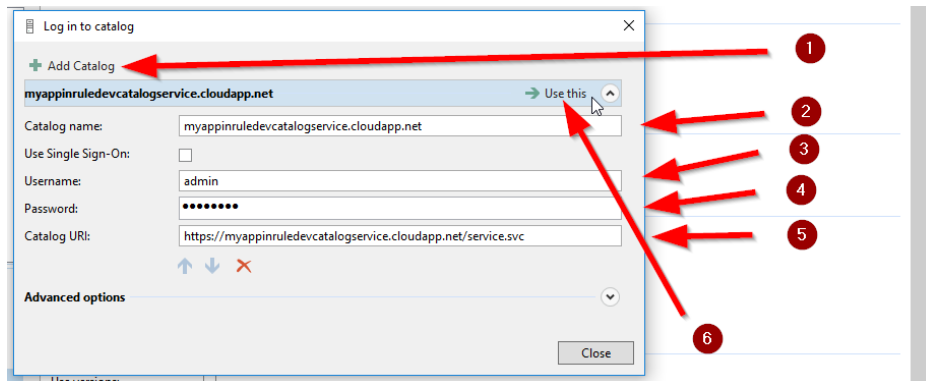
1: Navigate to the file in your file explorer and double click on it, this will open the file with irAuthor.



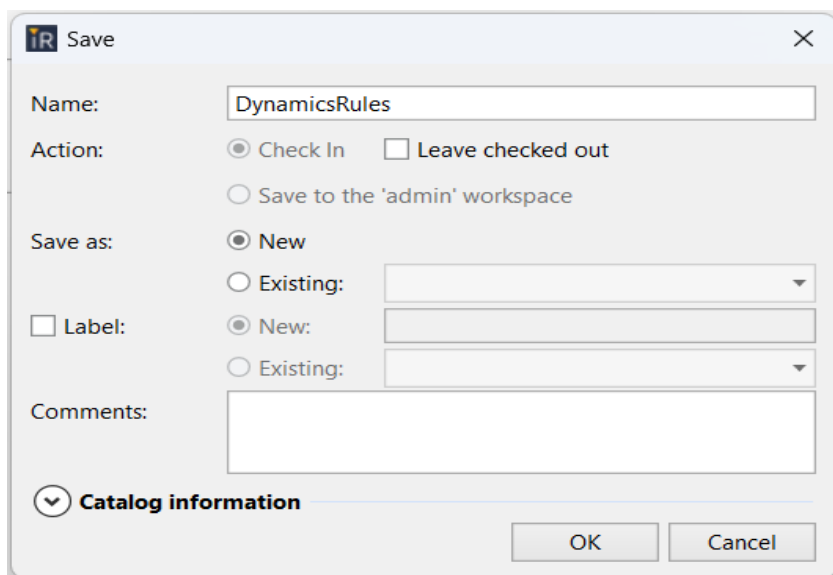
2: Save the Rule Application by choosing File ☐ Save As ☐ Save to Catalog



3: Choose Add Catalog, enter connection information for the Catalog Server that you deployed, and then select Use This.



4: Save with the name DynamicsRules



Save the Rule Application to the Catalog using the name of *DynamicsRules*.

At this point, if you can click OK without an error, we have successfully saved the DynamicsRules Rule App to the new Catalog that you have created. We can now continue by creating the Relay Namespace.

If you have any trouble getting to this point, it is advised that you resolve any issues with the Catalog before attempting to continue.

3.3.2 Rule Execution App Service for Dynamics 365

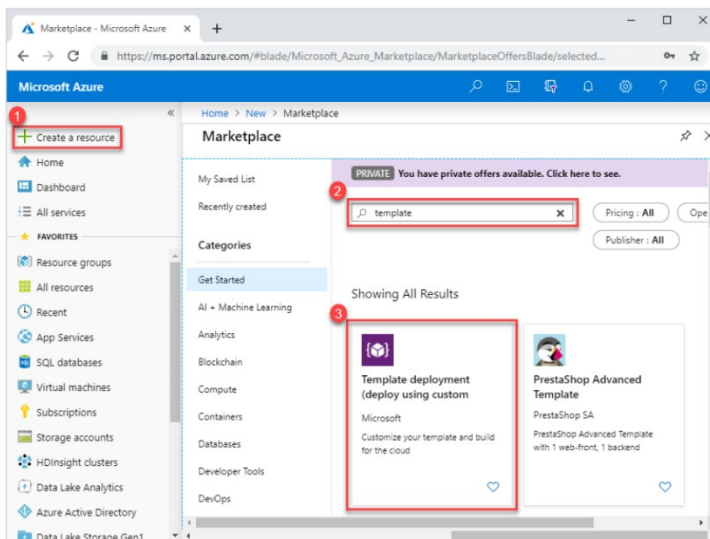
This section is only required for those following the ARM Template Deployment Path.

Next, we will deploy the InRule Rule Execution service as an Azure WebJob running on an App Service, along with all its Azure resource dependencies. To make this process easier, we'll be using an Azure Resource Manager (ARM) template, which allows us to deploy and configure all the Azure resources the Rule Execution Service relies on.

There are a number of methods for deploying an ARM template; this documentation will detail two: via **Azure CLI** and via **PowerShell**.

Alternatively, while this document does not provide a walkthrough of it, the ARM template provided is configured to work with **Azure Portal** deployment. For an overview of how to leverage ARM deployment through the Azure Portal, reference Microsoft's documentation: [Deploy resources with ARM templates and Azure portal](#) and navigate to the section titled "Deploy resources from custom template".

You can navigate directly to the Azure Portal page for deploying an ARM template at this link: [Custom deployment - Microsoft Azure](#).

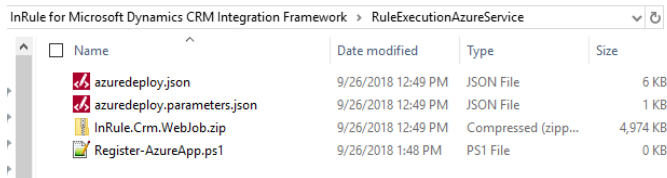






1: Locate azuredeploy.parameters.json

Before deploying the ARM template, we need to define certain parameters.

The required *azuredeploy.json* and *azuredeploy.parameters.json* files can be downloaded here - [AzureAppServices/Dynamics at master · InRule/AzureAppServices · GitHub](#).

Alternatively, they can be located within the *InRule for Microsoft Dynamics 365 Integration Framework.zip* file downloaded in [Section 3.2: Optional Resource Files](#).




InRule for Microsoft Dynamics CRM Integration Framework > RuleExecutionAzureService				
<input type="checkbox"/> Name	Date modified	Type	Size	
 azuredeploy.json	9/26/2018 12:49 PM	JSON File	6 KB	
 azuredeploy.parameters.json	9/26/2018 12:49 PM	JSON File	1 KB	
 InRule.Crm.WebJob.zip	9/26/2018 12:49 PM	Compressed (zipp...	4,974 KB	
 Register-AzureApp.ps1	9/26/2018 1:48 PM	PS1 File	0 KB	

2: Update parameters

Open the file with your text editor of choice and edit the parameters listed below

```
{
  "$schema": "https://schema.management.azure.com/schemas/2015-01-01/deploymentParameters.json#",
  "contentVersion": "1.0.0.0",
  "parameters": {
    "appServiceName": {
      "value": ""
    },
    "relayName": {
      "value": ""
    },
    "catalogUri": {
      "value": ""
    },
    "catalogUser": {
      "value": "admin"
    },
    "catalogPassword": {
      "value": "password"
    },
    "ruleApplabel": {
      "value": ""
    },
    "crmConnectionString": {
      "value": ""
    },
    "appServicePlanName": {
      "value": ""
    },
    "createOrUpdateAppServicePlan": {
      "value": true
    },
    "inRuleVersion": {
      "value": "5.7.3"
    },
    "appInsightsInstrumentationKey": {
      "value": ""
    },
    "appInsightsConnectionString": {
      "value": ""
    },
    "appInsightsResourceName": {
      "value": ""
    }
  }
}
```

1. appServiceName	Provide a name for the Azure App Service that the Rule Execution Service will run on.
2. relayName	Provide a name for the Azure WCF Relay (aka Azure Relay). This is NOT the desired namespace URL for the Relay, it should be the desired name of the actual Azure resource.
3. catalogUri	The URI for the Catalog Service that will be used Example: https://myappinruledevcatalogservice.cloudapp.net/service.svc

4. catalogUser	Username for Catalog Service, default value is 'admin'.
5. catalogPassword	Password for Catalog Service, default is 'password', please change this using the catalog manager!
6. ruleAppLabel	Optionally supply a default label used by the execution service to identify the ruleapp version to run.
7. crmConnectionString	<p>Provide a Dynamics connection string of type OAuth to use a regular named account and type ClientSecret for S2S authentication. Information on connection string formatting can be found here:</p> <p>https://docs.microsoft.com/en-us/powerapps/developer/common-data-service/xrm-tooling/use-connection-strings-xrm-tooling-connect</p> <p>For OAuth authentication, you can use the default app provided by Microsoft instead of creating and configuring your own application. You can find more about this, as well as the transition from WS-Trust (Office 365) to OAuth authentication here:</p> <p>https://docs.microsoft.com/en-us/powerapps/developer/data-platform/authenticate-office365-deprecation</p> <p>If using a named account, it should have the 'InRule Integration Administrator' role, or a custom role with read access to the entities used to run rules. It is important to note that the 'InRule Integration Administrator' role is not updated when new entities are created. Reference Appendix M: Known Issues, Limitations and Troubleshooting for more information.</p> <p> Important: Some customers have reported needing to provide a username in the format "domain\username" in order to successfully connect using IFD.</p>
8. appServicePlanName (If you wish to override the default value)	<p>This is an optional parameter, that, if left blank, will result in the ARM Template creating an AppServicePlan for you, using your defined appServiceName and appending "Plan" to the end. For example, defining your appServiceName as "ExampleAppService" would yield the following automatically generated App Service Plan name: "ExampleAppServicePlan."</p> <p>This parameter is intended to be used if you wish to either have a new AppServicePlan be created using a different name than the one outlined above, or for defining the name of a pre-existing App Service Plan that you would like to create your resources under. In the latter case, additional steps are required. For a more comprehensive guide on how to deploy your resources under an existing AppServicePlan, refer to Appendix H: Azure App Service Plan Configuration</p>
9. createOrUpdateAppServicePlan	By default, this value is set to true, and an App Service Plan will be created by the ARM template or updated if it already exists using the default naming convention. If you wish to deploy your app service to an already existing App Service Plan rather than create a new one, set this value to false and reference Appendix H: Azure App Service Plan and Application Insights Configuration .
10. inRuleVersion (To deploy most modern version, leave as default value)	This parameter allows the user to configure what version of the InRule Rule Execution Service they wish to deploy. By default, this parameter will be set to the most modern version.
11. appInsightsInstrumentationKey (If you wish to use an existing AI resource)	Provide an Instrumentation Key if you have an existing App Insights resource you'd like to use for logging and telemetry. If you are configuring this in a nonstandard azure environment (such as Azure Government), please additionally provide an App Insights Connection String. Otherwise, leave this value blank and provide a value for the 'appInsightsResourceName' parameter, which will create the resource for you. For more information on the logging view Rule Execution Service Event Log .
12. appInsightsConnectionString (If you wish to use an existing AI resource in a non-standard Azure environment)	<p>Only override the default value here if you have an existing App Insights resource you'd like to use, AND you need to use a non-standard connection string. If you need to supply your own connection string, be sure to set that value here, as well as providing your instrumentation key in the appInsightsInstrumentationKey parameter. If you want this template to manage the App Insights resource for you, or only need to provide an instrumentation key, leave this value as the default and provide values for appInsightsResourceName or appInsightsInstrumentationKey instead.</p> <p>Additionally, for any consideration about using app insights or setting it up in a non-standard Azure environment view Appendix I: Azure App Insights Configuration</p>

13. applnsightsResourceName
(If you wish to create a new AI resource)

If you want to use Application Insights (AI) as a log sink in addition to the app service logging already enabled, but do **not** already have an Insights resource that you want to use, specify a name for a new resource here. Specifying a value for this parameter will create a new Application insights resource with the given name and populate the instrumentation key app setting on the app service with the key from this new resource. **If you provide a value for this parameter, do not provide a value for applnsightsInstrumentationKey.** If you receive an error about the location when deploying the template, refer to [Application Insights Location Error](#)

Once you've finished configuring your parameters, save the completed parameters file and keep a spare copy on hand for future upgrades or automation.

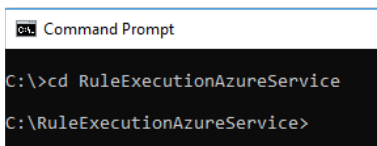
3: Option 1: Deploy ARM Template with Azure CLI

Now that the ARM template is configured, we'll deploy it to get the resources up and running. The following will detail how to use the Azure CLI to deploy the ARM template (Note, this section assumes Azure CLI has already been installed):

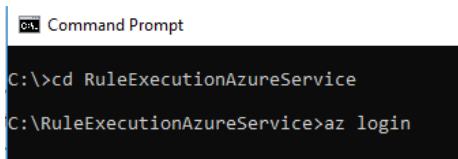
3.1 Run Command Prompt or Powershell



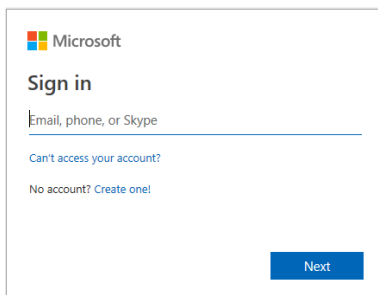
3.2 Navigate to the RuleExecutionAzureService folder



3.3 Enter “az login” to login into Azure



3.4 Enter your Azure admin credentials to login when prompted in the new browser window opened



3.5 Select the appropriate subscription

If your Azure account has access to multiple subscriptions, you will need to set your active subscription to where you create your Azure resources:

```
C:\RuleExecutionAzureService>az account set --subscription SUBSCRIPTION_NAME
```

3.6 Create Resource group

If you have not created a resource group yet, you will need to create one. You will need to define a name and a geographic location for where to host the resource. This example uses Central US:

```
C:\RuleExecutionAzureService>az group create --name ResourceGroupName --location centralus
```

3.7 Execute the following command to deploy the ARM template

Replace “ResourceGroupName” with the name of the Azure Resource Group you want to deploy to

```
C:\RuleExecutionAzureService>az deployment group create -g ResourceGroupName  
--template-file .\azuredeploy.json --parameters .\azuredeploy.parameters.json  
--query properties.outputs.replayKey.value -o table
```

Observe that the template deploys with no errors

If the template deploys successfully, you will see a result that looks similar to this:

```
Result  
-----  
kwSQxK+
```


The value that is output is your Azure Relay key, which is a value you will need to configure your Dynamics instance in a later step.

4: Option 2: Deploy ARM Template with Powershell


(If you have already deployed the ARM template via Azure CLI in the section above, this section is not necessary)

Now that the ARM template is configured, we'll deploy it to get the resources up and running. The following will detail how to use Powershell to deploy the ARM template (Note, this section assumes Azure PowerShell has already been installed):

4.1 Run Powershell

 Windows PowerShell

4.2 Navigate to the RuleExecutionAzureService folder

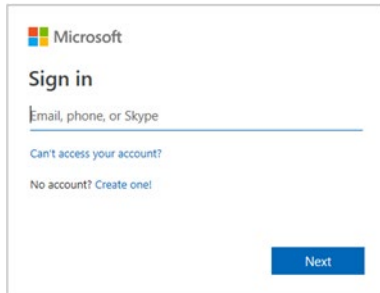
 Windows PowerShell

```
PS C:\> cd .\RuleExecutionAzureService\  
PS C:\RuleExecutionAzureService>
```

4.3 Enter “Connect-AzAccount” to login into Azure

```
PS C:\RuleExecutionAzureService> Connect-AzAccount
```

4.4 Enter your Azure admin credentials to login when prompted in the new browser window opened



4.5 Select the appropriate subscription

Upon logging in, your default subscription information will be displayed:

```
> Administrator: Windows PowerShell

Account      : 
SubscriptionName : 
SubscriptionId : 
TenantId     : 
Environment  :
```

If this is not the subscription you want to deploy to, you can use the “Select-AzureRmSubscription” cmdlet to change the targeted subscription. Just replace “SubscriptionNameHere” with the name of the desired subscription:

```
PS C:\RuleExecutionAzureService> Select-AzSubscription -SubscriptionName SubscriptionNameHere
```

4.6 Create Resource Group

If you have not created a resource group yet, you will need to create one. You will need to define a name and a geographic location for where to host the resource. This example uses Central US:

```
PS C:\RuleExecutionAzureService> New-AzResourceGroup -Name ResourceGroupName -Location centralus
```

4.7 Execute the following command to deploy the ARM template

Replace “ResourceGroupName” with the name of the Azure Resource Group you want to deploy to

```
PS C:\RuleExecutionAzureService> New-AzResourceGroupDeployment -ResourceGroupName ResourceGroupName
-TemplateFile .\azuredploy.json -TemplateParameterFile .\azuredploy.parameters.json
```

Observe that the template deploys with no errors

If the deployment is successful, you should see an output similar to this:

DeploymentName	: azuredeploy	
ResourceGroupName	: [REDACTED]	
ProvisioningState	: Succeeded	
Timestamp	: 9/27/2018 7:45:26 PM	
Mode	: Incremental	
TemplateLink	:	
Parameters	:	
	Name	Type
	Value	
	[REDACTED]	
Outputs	:	
	Name	Type
	Value	
	relayKey	String
	[REDACTED]	

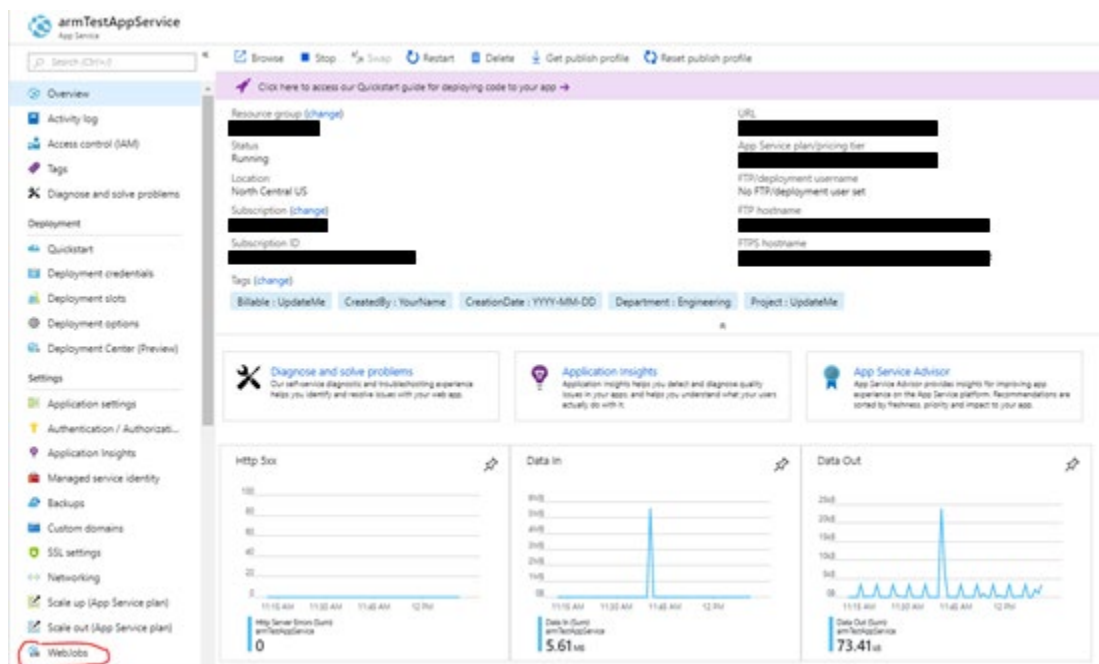
Note the value of “relayKey” underneath the “Outputs” heading. This value is your Azure Relay key and will be needed when configuring your Dynamics instance in later steps.

5: Verify Setup

Navigate to the Azure portal and locate the deployed App Service

1 of 2 items selected	<input type="checkbox"/> Show hidden types
<input checked="" type="checkbox"/> NAME	TYPE
<input checked="" type="checkbox"/> armTestAppService	App Service

Click on the app service, and on the nav-bar that appears to the left of the resource overview, select WebJobs



Ensure the InRule.Crm.WebJob is present and its “Status” is set to “Running”

**WebJobs**

WebJobs provide an easy way to run scripts or programs as background processes in the context of your app.

NAME	TYPE	STATUS	SCHEDULE
InRule.Crm.WebJob	Continuous	Running	n/a

6: Upload License File

Regardless of how you choose to deploy the ARM template, you'll need to upload a license file to it for the rule execution app service to properly function. The simplest way to upload the license file is via the Azure App Service Editor. Alternatively, you can deploy the license file via FTP. A walkthrough of how to upload the license file by either of these methods can be found in [Appendix K: License Management](#).

3.3.3 InRule Solution for Dynamics 365

At this point, all of the Azure Requirements are met. The Azure Relay should be listening for incoming communication from Dynamics. We must now setup Dynamics, which can be done with either an **AppSource Deployment** or a **PowerShell Deployment**. If trying to deploy a previous version of the Dynamics solution, the PowerShell Deployment must be used.

- **AppSource Deployment** -- You can install the InRule Solution through Microsoft AppSource. The InRule listing can be found here in the [Microsoft AppSource](#). Once the solution is deployed, you can proceed to [Configuring the InRule Solution in Dynamics](#)
- **PowerShell Deployment** -- You can follow the guide below to deploy the Dynamics package via PowerShell script. This is required if deploying a previous version of the Dynamics solution and relies on the resources from the Integration Framework zip file outlined in the section [Optional Resource Files](#)



Important: You only need to deploy the InRule Solution by **one** of the two above methods. Doing both is unnecessary.



Important: Only deploy the solution using one of the two above methods. Do not manually import included solution files.

1: Navigate to the '\DynamicsDeployment' directory:

```
Administrator: Windows PowerShell
PS C:\> cd .\DynamicsDeployment\
PS C:\DynamicsDeployment>
```

2: Execute Deploy-CrmPackage.ps1

Deploy-CrmPackage.ps1

```
PS C:\> .\Deploy-CrmPackage.ps1
```

If you are using S2S authentication in the execution service, you will need to pass in the Azure Active Directory application ID as the “AzureAppId” parameter to the Deploy-CrmPackage.ps1 script. This is the application ID that was output in [Section 3.2: Registering an Azure Active Directory Application](#). It will be used to create a new Application User in Dynamics. If you were not involved in registering your Azure Active Directory application, your Azure administrator should be able to provide you with this ID. If you are using connection-string based authentication in the execution service, you do not need to provide this parameter.

```
PS C:\DynamicsDeployment> .\Deploy-CrmPackage.ps1 -AzureAppId <app ID here>
```

Additionally, the script supports 2 other parameters: “SbNamespaceAddress” and “SasKey”. These parameters allow you to pass in the **fully qualified** Relay Namespace Address (e.g.: https://<<RelayNamespace>>.servicebus.windows.net/<<SB Relay Path>>) and Relay Key for the Relay your Rule Execution App Service is listening on and will allow the script to auto-configure your Dynamics instance to point at that Relay. These parameters operate independently of the “AzureAppId” parameter; they may be alongside it or on their own. However, if you opt to provide a value for either “SbNamespaceAddress” or “SasKey,” you must provide a value for the other as well.

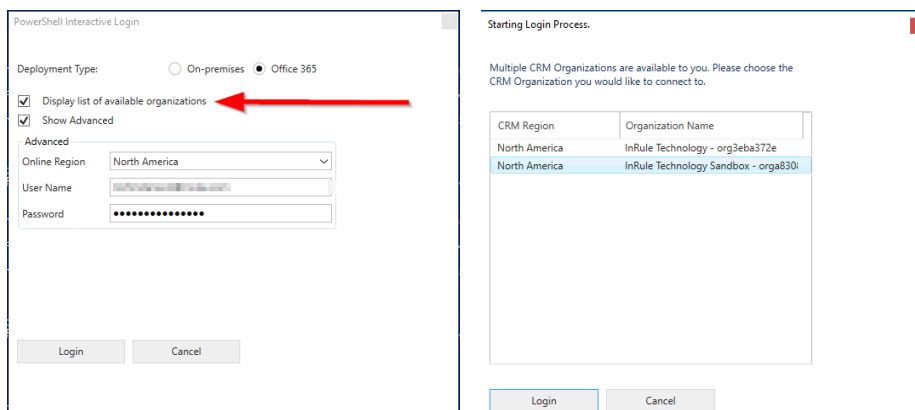
```
PS C:\DynamicsDeployment> .\Deploy-CrmPackage.ps1 -SbNamespaceAddress <service bus name here> -SasKey <service bus key here>
```

If you choose not to provide the “SbNamespaceAddress” and “SasKey” parameters, you will receive a notification (shown below) after the script finishes executing reminding you that you need to manually configure the SAS Namespace and SAS Key as outlined in [Step 4](#) of this section.

The SAS Key needs to be set in the Rule Execution Service Endpoint Configuration in order to run rules

3: Provide Dynamics 365 Service Credentials

By default, the script will display an interactive login window to connect to Dynamics. Please be sure to check the ‘Display list of available organizations’ to make sure that you select the correct instance of Dynamics to install to!



Alternatively, the Deploy-CrmPackage.ps1 script also accepts a Dynamics Connection String:

```
.\Deploy-CrmPackage.ps1
```

```
-CrmConnectionString 'AuthType=Office365;Url=https://irroadsandbox.crm.dynamics.com/;Username=CRM;Password=;'
```

```
.\Deploy-CrmPackage.ps1 -CrmConnectionString  
'AuthType=Office365;Url=https://{DYNAMICSURL}.crm.dynamics.com/;Username={DYNAMICSUSERNAME};Pa  
ssword={DYNAMICSPASSWORD}'
```

Observe that no errors occur while the Deploy Script is running.



Important: This deployment can take a long time to complete. In most cases it takes around 20 minutes.

4: Configure the InRule Solution in Dynamics

Next, we need to configure the Dynamics Solution that has been deployed to point to the Rule Execution Service that is already available on Azure. We will need to have the Relay Namespace and the Relay Key (SAS Key) ready for this step. You can find more information about these settings in the [Rule Execution App Service for Dynamics 365](#) section.

Navigate to the Advanced Settings for your organization, then the Rules Configuration page under the InRule group:

The screenshot shows the Microsoft Dynamics 365 Settings interface. The top navigation bar includes 'Dynamics 365', 'Settings', and 'Business Management'. The main content area is divided into several sections: Business, Customization, System, Process Center, InRule, and Extensions. The InRule section is highlighted, and the 'Rules Configuration' link is selected. Below this, the 'Configure InRule for Microsoft Dynamics CRM' page is displayed. This page contains two main configuration panels: 'Service Endpoint Configuration' and 'Step Registration Configuration'. The 'Service Endpoint Configuration' panel includes fields for 'Service Endpoint ID' (d2e50ca1-e4fc-e611-80e9-6c3be5a82b30), 'Endpoint Name' (Default InRule Service Endpoint), 'SAS Key Name' (RootManageSharedAccessKey), 'SAS Key' (with a 'Change' button), 'Azure SB Namespace Address' (https://namespace.servicebus.windows.net/ruleexecution), and 'SB Relay Path' (ruleexecution). There are 'Save' and 'Test' buttons at the bottom of this panel. The 'Step Registration Configuration' panel has a 'Select an InRule plugin step' dropdown and an 'Add New' button. At the bottom of the page, there is a 'Rule Configurations' section.

You will need to define values in the following fields.

Configuration Form Fields	Details on these settings are found in “ Rule Execution App Service for Dynamics 365 ” section.
Endpoint Name	For basic configuration, this can be left as is.
SAS Key Name	The key output by the ARM template is the ‘RootManageSharedAccessKey’ key, so this value can be left as the default in most scenarios.
SAS Key	This is the key secret value, and is output by the ARM template after completion of the deployment. If you did not get the value during deployment, you can find it using the steps in Locate SAS Key in Azure Portal Note: If you provided the “SasKey” parameter in Step 2 of this section, this value will already be set.
Azure SB Namespace Address	Change the Relay namespace here, as the default value of ‘inrule-crmonline’ will not work for you. The relay namespace is the same value provided for the relay name in the ARM template parameters file. Using the Relay Namespace and the Relay Path, construct the address as follows: https://<<RelayNamespace>>.servicebus.windows.net/<<SB Relay Path>> Note: If you provided the “SasName” parameter in Step 2 of this section, this value will already be set.
SB Relay Path	Unless you have changed the corresponding setting in rule execution app service, this value can be left as the default, which is ‘ruleexecution’

The following screenshots show the Service Endpoint Configuration screen before and after configuration.

- [#1 in screenshot] SAS Key Name
- [#3 in screenshot] SAS Key
- [#4 in screenshot] Azure SB Namespace Address
- [#5 in screenshot] SB Relay Path

Please note that you must click on ‘SET’ [#2 in screenshot] before the SAS Key can be entered.

For a more complete description of all of the configuration options available, reference [Appendix F: Rules Configuration and Settings](#)

Notice: If you are using the standard Azure US environment ‘AzureCloud’, then the provided domain suffix will work fine. However, if you are using an alternate Azure Environment such as US Government or an international environment – be sure to update the suffix to the appropriate value for your region.

The left screenshot shows the 'Service Endpoint Configuration' form with the following fields and values:

- Service Endpoint ID: d2e50ca1-ef4c-e611-80e9-6c3be5a82b30
- Endpoint Name: Default InRule Service Endpoint
- SAS Key Name: RootManageSharedAccessKey
- SAS Key: Set
- Azure SB Namespace Address: https://inrule-crmonline.servicebus.windows.net/ruleexecution
- SB Relay Path: ruleexecution

The right screenshot shows the same form after saving, with the following changes:

- SAS Key Name: Irrodsandbox2crm2-3-4-75servicebusAuthKey
- SAS Key: Cancel
- Azure SB Namespace Address: https://irrodsandbox2crm2-3-4-75servicebus.servicebus.windows.net/ruleexecution
- SB Relay Path: ruleexecution

Be sure to click save, and to observe the “Saved Changes” notice:

A green message box with the text "Saved changes" and a close button (X).

Once changes have been saved, configuration can be tested by click the test button [#7 in screenshot]. This test configuration tests these three things:

A screenshot of the test results showing three successful connections:

- Test Execution Service Connection: Success
- Test Execution Service to Dynamics Connection: Success
- Test Rule Catalog Connection: Success

1. Execution Service Connection. Test sends a request to the execution service that was deployed and configured to make sure that it is setup correctly.
2. Execution Service to Dynamics Connection. Test makes a request from the execution service to make sure that it can make a request back to Dynamics with the connection information configured on the App Service.
3. Rule Catalog Connection. Test makes a request to the Rule Catalog with the connection information configured on the App Service.

If all the tests succeed, the system has been configured correctly. If any of the tests fail, check the configurations for the subject under test.

5: Configure S2S App User

Dynamics 365 supports S2S authentication via Azure AD app as an alternative to username/password authentication. This removes the need to create a licensed user, and avoids tying authentication to a regular named account. If you are using InRule SaaS, this Azure AD app has already been created and configured for you, but if you are self-hosting you can create your own Azure AD app as outlined in [Registering an Azure Active Directory Application \(Optional\)](#) and use it to authenticate.

Whichever method you are using, you will need to create an associated 'Application User' inside Dynamics. When configuring a InRule SaaS setup, you can follow the instructions in [Appendix I: InRule SaaS Portal Configuration](#), which will create and set up the Application User for you. If you are self-hosting, use the steps below.



Important: These steps are only necessary for users that want to leverage S2S authentication and are self-hosting. If you are self-hosting, or using username/password OAuth authentication, you do not need to follow any of the steps below

5.1: [Sign-in to the Power Platform admin center on a system admin account](#)

5.2: Select Environments, and then select an environment from the list

Environment	Type	State
InRule Technology QA Sandbox 2	Sandbox	Ready
InRule Technology QA Sandbox	Sandbox	Ready

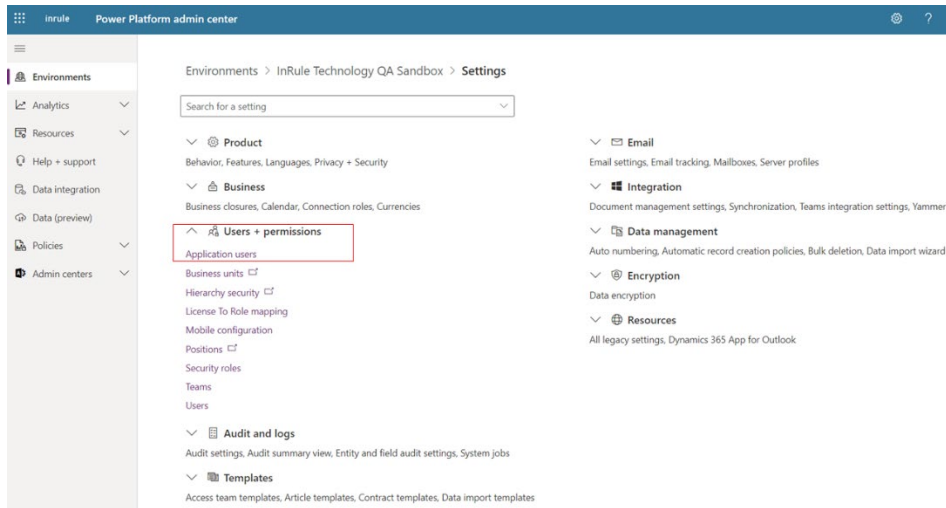
5.3: Select Settings

Details	
Environment URL irqasandbox.crm.dynamics.com	State Ready
Region United States	Refresh cadence Moderate
Type Sandbox	Security group SEC Product Quality Assurance Dynamics Sandbox
Organization ID dd803915-6832-441a-877a-6644583022bd	

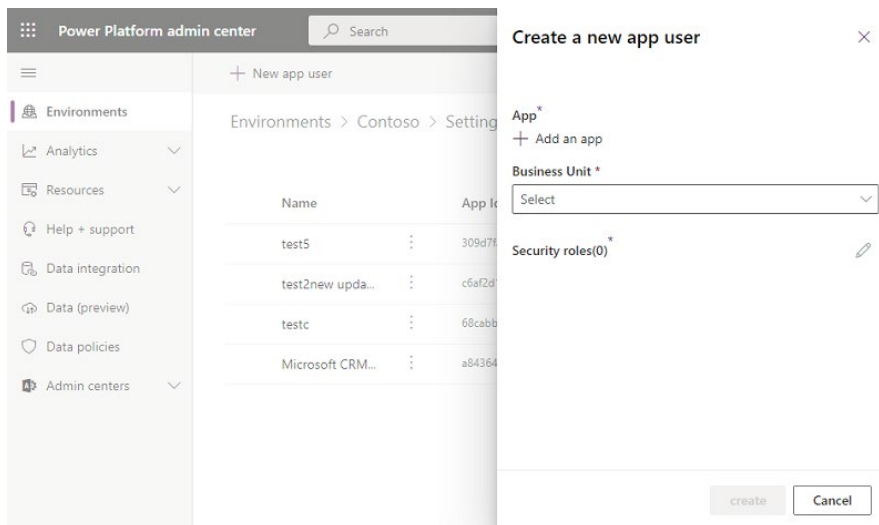
Access

- Security roles**
[See all](#)
- Teams**
[See all](#)
- Users**
[See all](#)
- S2S Apps**
[See all](#)

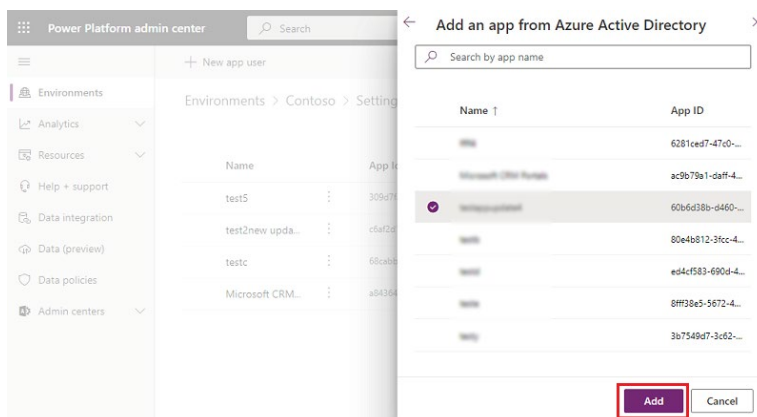
5.4: Select Users + permissions, and then select Application users



5.4: Select + New app user to open the Create a new app user page



5.5: Select Add an app to choose the registered Azure AD application and then select Add



5.6: Under Business Unit, select a business unit from the dropdown list

Create a new app user ✕

App ^{*}
inruleSample ✎

Business unit ^{*}

irqasandbox ▾

Security roles(0) ✎

5.7: Select Edit for Security roles

Create a new app user ✕

App ^{*}
inruleSample ✎

Business unit ^{*}

irqasandbox ▾

Security roles(0) ✎

5.8: Tick the box for "InRule Integration Administrator" role and click Save

<input type="checkbox"/>	Help Page Author
<input type="checkbox"/>	Help Page Consumer
<input checked="" type="checkbox"/>	InRule Integration Administrator
<input type="checkbox"/>	InRule Rule Executor
<input type="checkbox"/>	IoT - Administrator
<input type="checkbox"/>	IoT - Endpoint User
<input type="checkbox"/>	Knowledge Manager
<input type="checkbox"/>	Marketing Manager
<input type="checkbox"/>	Marketing Professional
<input type="checkbox"/>	Office Collaborator
<input type="checkbox"/>	Orchestrator
<input type="checkbox"/>	Playbook Manager
<input type="checkbox"/>	Playbook User

Save

Cancel

5.9: Click Create

Create a new app user

App *

inruleSample

Business unit *

irgasandbox

Security roles(1)

InRule Integration Administrator

Create Cancel

6: Verify a successful deployment!

A 'Run Rules' button will now exist on the edit form of all Dynamics entities. The way this button behaves will depend on the settings that have been configured under the Rule Configuration section; this document assumes the default configuration behavior. To understand what settings are available and what they mean, please see [Appendix F: Rules Configuration and Settings](#) of this document.

Open up an Account and execute 'Run Rules'. The default deployment is configured to run a Rule App from the catalog named DynamicsRules, which is what we uploaded to the catalog in the [Verify Catalog](#) stage. If everything has been set up correctly, you should see the "Rule Execution Completed" message and the description of the account will be updated to provide the date and time.

7: Getting Users Ready

While the parties responsible for deploying out InRule can likely leverage the Run Rules button without issue as they're likely to be system administrators, if your organization intends to empower non-admin users to be able to leverage the Run Rules button, they'll encounter permission issues. This is due to rule execution requiring access to various Dynamics configuration entities that normal users don't generally have permissions to touch.

To circumvent this, the InRule Solution for Dynamics ships with a pre-configured role called the "InRule Rule Executor," which contains the bare minimum permission set necessary for users to be able to make use of the Run Rules button. You'll need to assign any users in your organization to this role if the intent is to empower them to run rules through the Dynamics UI.

The screenshot shows the Microsoft Dynamics 365 interface for the 'Blue Yonder Airlines (sample)' account. The top navigation bar includes 'Dynamics 365', 'Service', and 'Accounts' tabs. Below the navigation bar is a toolbar with buttons like 'NEW', 'RUN RULES', 'DEACTIVATE', 'CONNECT', 'ADD TO MARKETING LIST', 'ASSIGN', 'EMAIL A LINK', 'DELETE', 'FORM', and '***'. The main content area displays the account details for 'Blue Yonder Airlines (sample)', including 'Annual Revenue: \$10,000.00', 'No. of Employees: 2,900', and 'Owner: CRM Service'. A yellow banner at the top of the account details section contains two messages: 'Blue Yonder Airlines (sample) Account has 1 Contacts.' and 'Rule execution completed.' Red arrows point from the 'RUN RULES' button to the first message, and from the 'Rule execution completed' message to the 'CONTACTS' section. The 'CONTACTS' section shows a list of contacts, including 'Sidney Higa (sample)' as the 'Primary Business Contact'. The 'ACCOUNT INFORMATION' section on the left lists details like 'Account Name', 'Phone', 'Fax', 'Web Site', 'Parent Account', and 'Ticker Symbol'.

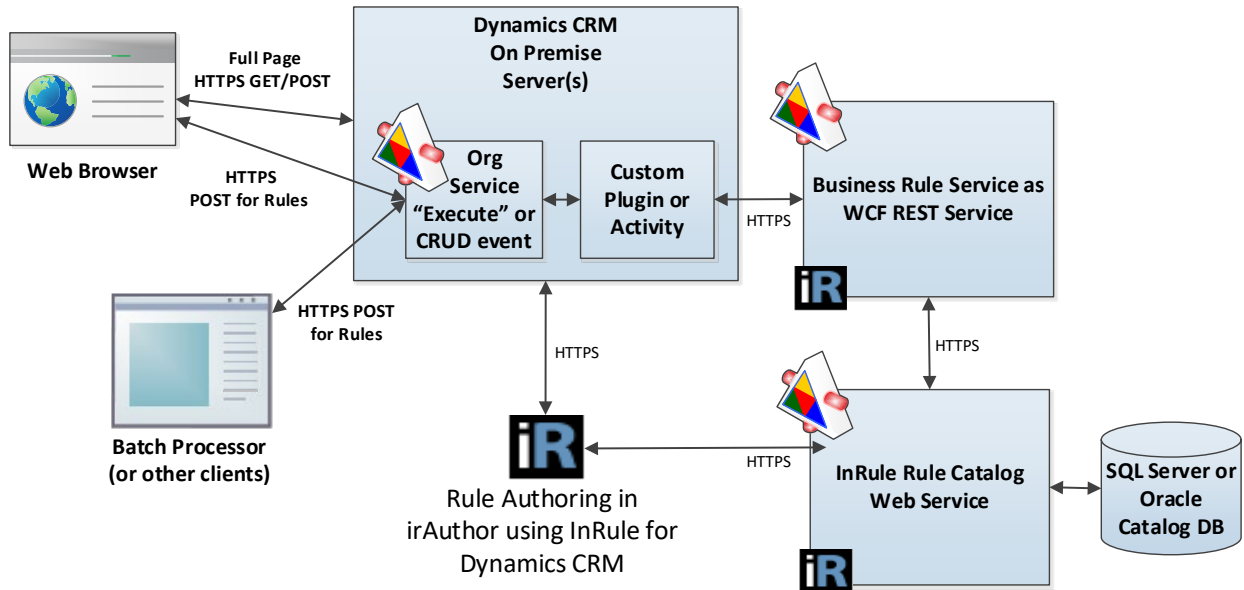
If there are any issues with executing rules and you are using an S2S user account from [Section 3.2](#), check [Appendix M: Known Issues, Limitations and Troubleshooting](#) of this document. If you are using the connection string, make sure the user account specified has the necessary privileges to access the particular entity.

4 Performing the Installation: On-Premises

This section discusses the steps needed to integrate InRule with Microsoft's on-premises version of Microsoft Dynamics.

This section assumes that you already have an IIS web server and an on-premises version of Dynamics deployed and configured.

4.1 An overview of the Components needed



Catalog Web Service and Database

A Catalog service will be used to store Rule Apps that will be consumed by the Rule Execution Service. This Catalog Service will be hosted as a web service on an IIS server. The back end of the Catalog Service utilizes a SQL Server or Oracle database for retrieval and persistence of Rule Applications. The Installation Documentation and InRule Installer, which will be used to install the service itself, are available on [our support website's downloads section](#) provides instructions for setting up the catalog.

Rule Execution Web Service for Dynamics 365 On-Premises

The WCF Web Service implementation is designed to be generically applied to any given Dynamics Entity and corresponding InRule rules that may apply to that Entity. After the web service is published, the service can be directed to run various Rule Applications against different Entity types by passing arguments in the REST calls. This will require a separate deployment than the Catalog Web Service; this process will be covered in the ["Deploy the Rule Execution Web Service"](#) section.

InRule Solution for Dynamics 365 On-Premises

The RuleExecutionServices solution contains a custom plugin, client resources, and a configuration form. Unlike the Online solution, this is configured to work directly against the execution service, rather than requiring the "middle-man" of the Azure Relay.

4.2 Gathering prerequisites

This section reviews what you will want to have prepared before you begin with the integration steps in the next section.

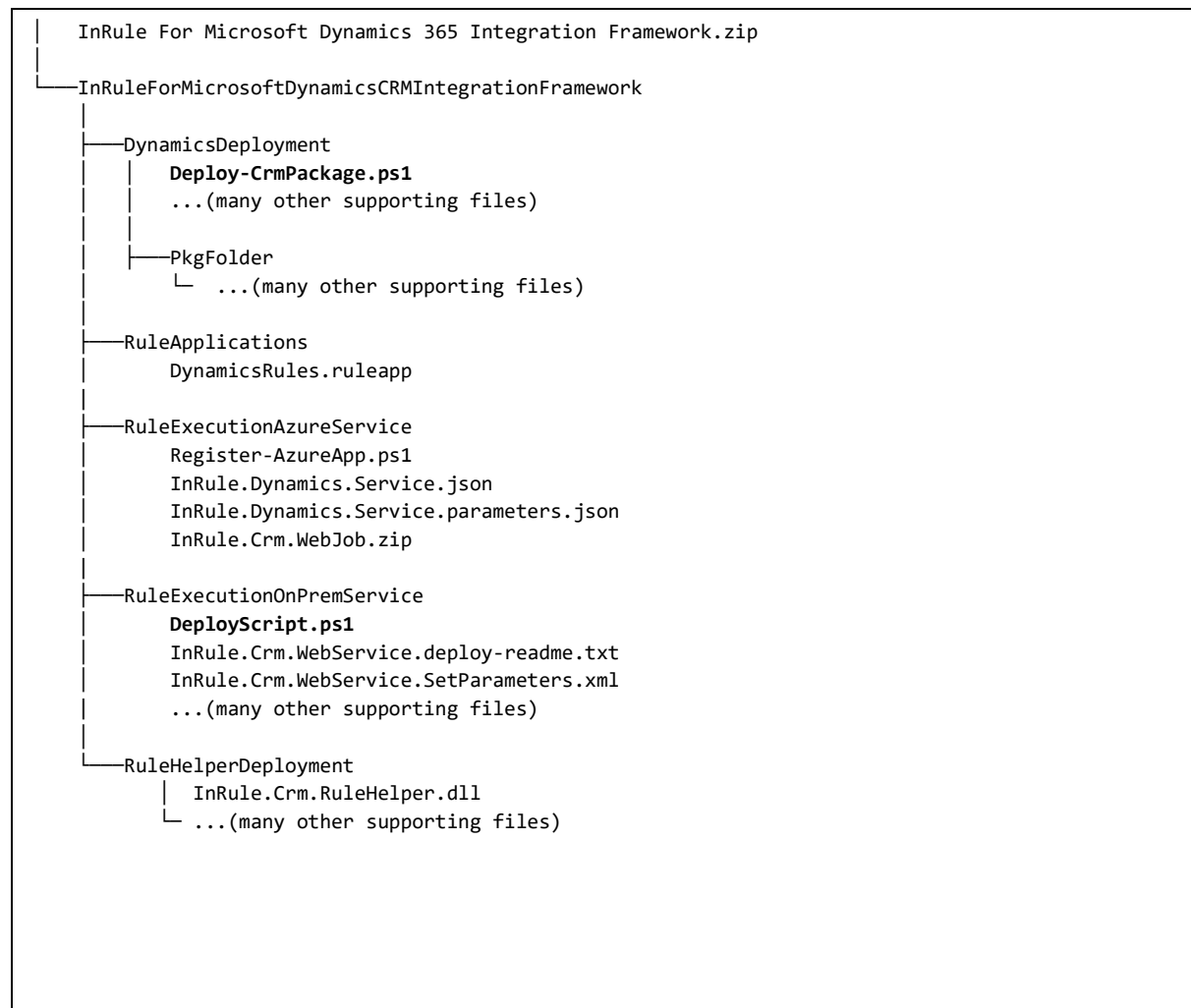
Required Files

The following files should be downloaded from [our support website's downloads section](#) before you begin:

- InRule for Microsoft Dynamics 365 Integration Framework.zip
- InRule Catalog Installer – InRule Installer.exe

Please download and extract each of these archives to corresponding subdirectories within a working folder.

When you are finished, you should have a directory structure that looks like this:



Files in **bold** will be used directly in the walkthrough steps below.

Rule Authoring Environment

A Rule Authoring Environment is used to upload a Rule Application to your catalog app service. A Rule Authoring Environment is a machine or virtual machine where irAuthor has been installed with the irX for Microsoft Dynamics 365 extension. If you followed the instructions outlined in [irX for Microsoft Dynamics 365 Help Documentation](#), then you should already have a rule-authoring environment available to you.

We have made it a point to call out the rule authoring environment separately because it is important to be aware of the licensing implications of this step. You will need to utilize an irAuthor license and an irX for Microsoft Dynamics 365 license for the duration of this process. If you're a system administrator who does not intend to perform rule authoring duties after the deployment is up and running, you can either choose to borrow an environment from someone who will use a rule authoring environment, or you will want to be sure to deactivate your license when you're finished with your deployment responsibilities.

Administrative Accounts

Dynamics On-Prem Organization Service URI: You will want to have the root URL of the organization web service exposed by your Dynamics 365 instance. The server URL is usually in the format of "[http://crm-server:port/organization-name](#)"

Dynamics Service Account Login and Password: You will want to have a username and account created specifically for use by the InRule for Microsoft Dynamics 365 Framework Web Service.

Administrative Password to use for SQL Server: You should decide what username and password you want to use for administrative privileges on the SQL Server. This walkthrough will construct a connection string that will utilize these resources to connect to the catalog.

*** This walkthrough will utilize the above administrative login and password for the Catalog Service to connect to the SQL Server Database. In a more secure environment, a separate SQL User should be created that only has access to the single database needed by the catalog. It is up to the reader of this document to go this more secure route.*

Administrative Password to use for Catalog Service: You should decide what username and password you want to use for administrative privileges within irCatalog.

*** This walkthrough utilizes the default login of 'admin' and password of 'password'. It will be up to the reader to go through the process of utilizing the Catalog Manager to change these credentials to be more secure.*

InRule License Keys

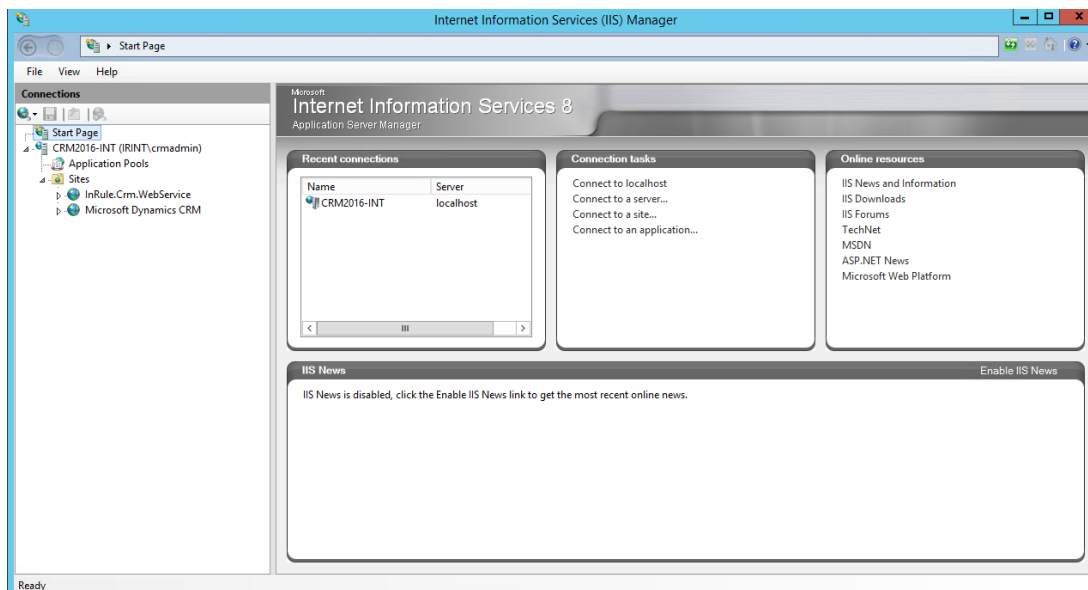
You will need your InRule irServer license keys. These can be found at <https://support.inrule.com/activations.aspx>. You can contact support@InRule.com if you have questions about where to get your license keys. For guidance on how to activate your keys, reference [Appendix K: License Management](#).

4.3 Deploying and Configuring Components

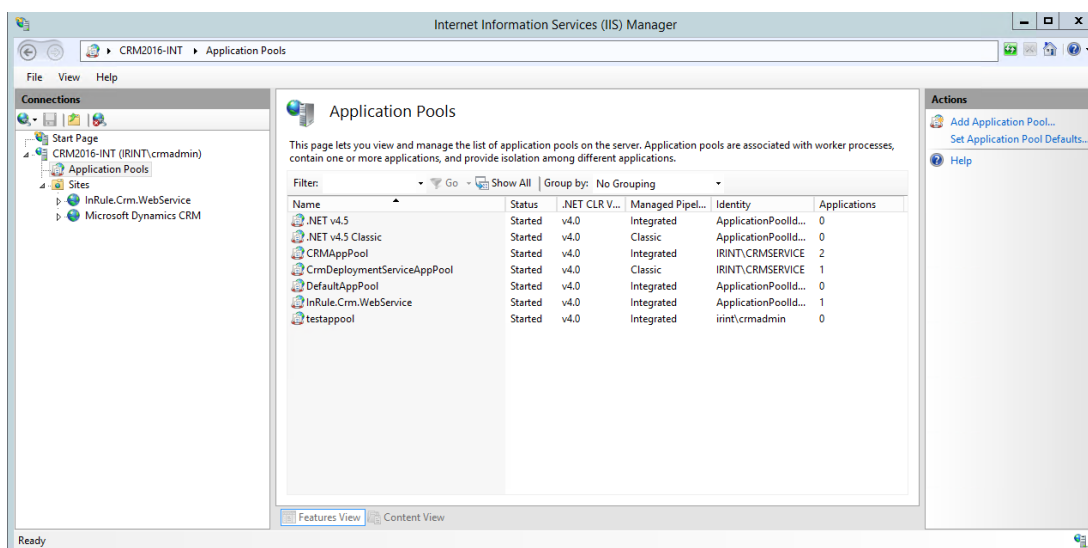
4.3.1 Setting Up an Application Pool in IIS

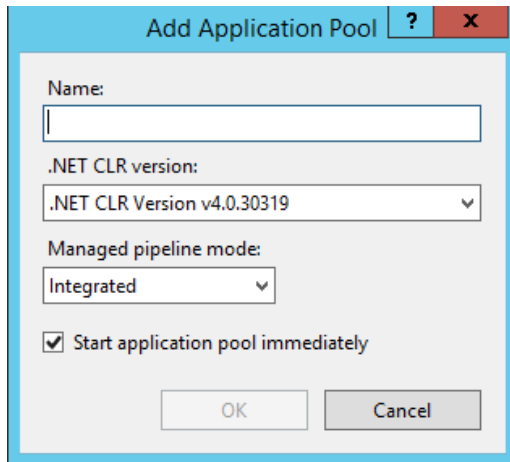
Before a website can be setup to host the catalog web service, a suitable application pool needs to be created. Any IIS server will already have a DefaultAppPool that can be used; whether this section is necessary will depend on your organizations architecture and requirements.

1: Open IIS Manager

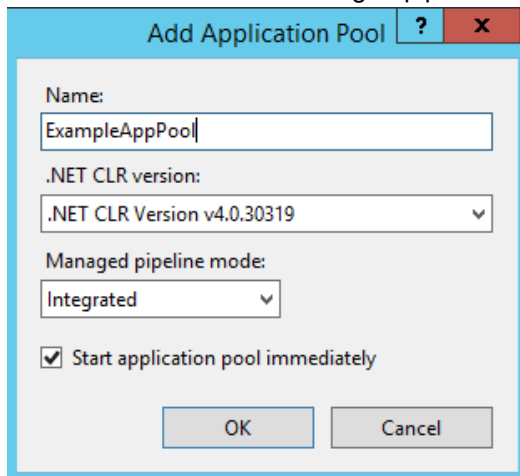


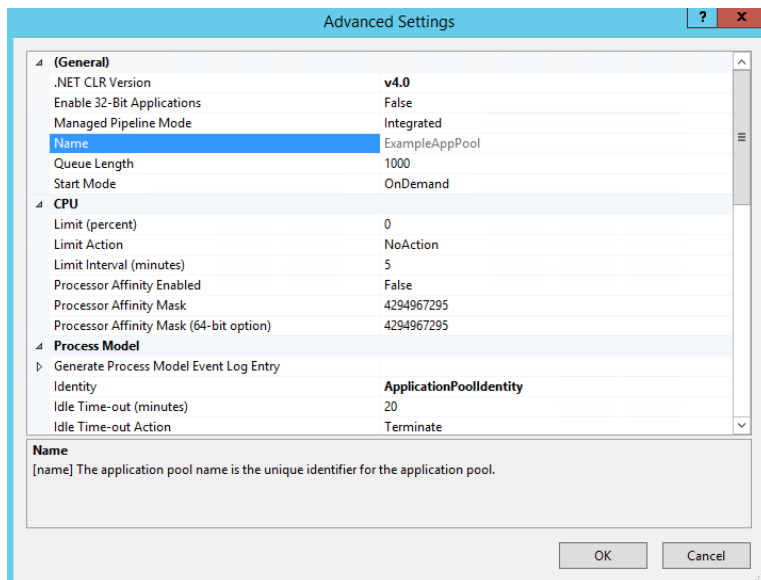
2: Click on Application Pools on the left-hand nav pane



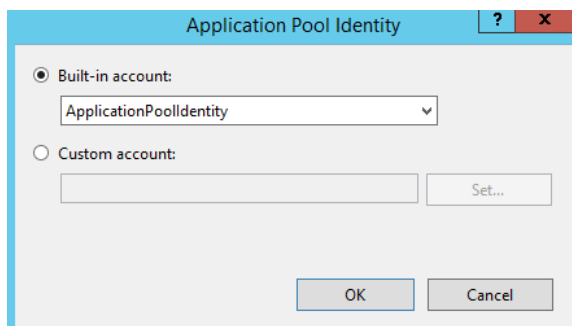
3: Click “Add Application Pool” under “Actions” in the right-hand pane**4: Define a name for the Application Pool and Create**

.NET CLR Version and Managed pipeline mode should be left as their default values. Press OK

**5: Click on the new App Pool and Select “Advanced Settings” on the right-hand pane**



6: Under the “Process Model” header, select the small “...” button next to the “Identity” property



Here you can select what kind of identity model you want to use for your application pool. You'll want to either use the Application Pool Identity or create a custom account.

The Application Pool Identity creates a virtual account with the same name as your new application pool. All worker processes within this application pool will run under this account

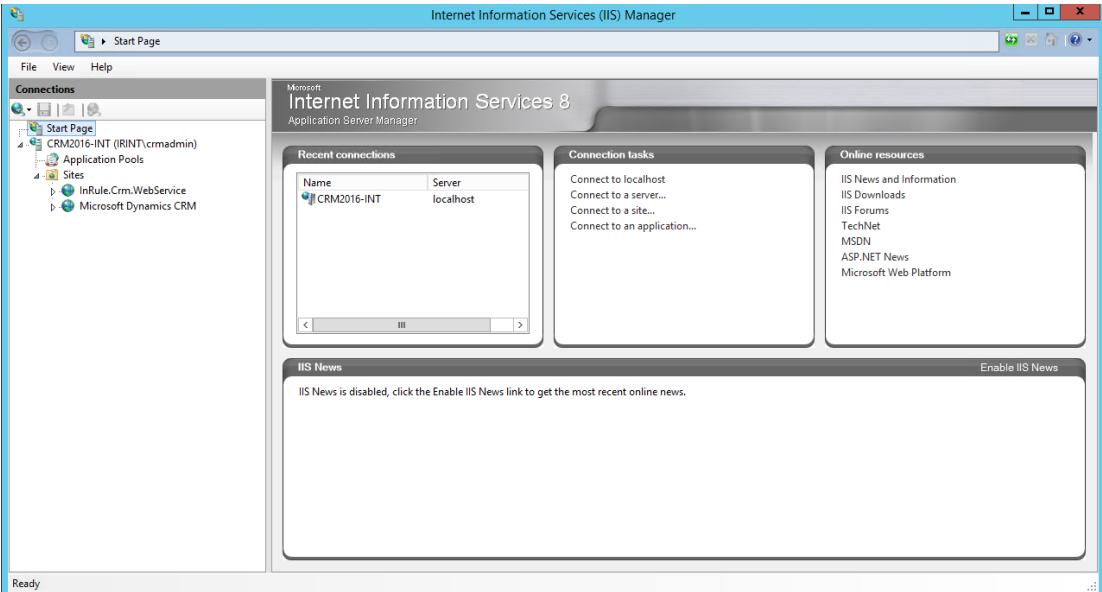
Custom accounts are the best solution if you want to use your Windows credentials to authenticate to the Catalog web service.

Press “OK” when you have made your selection and are finished.

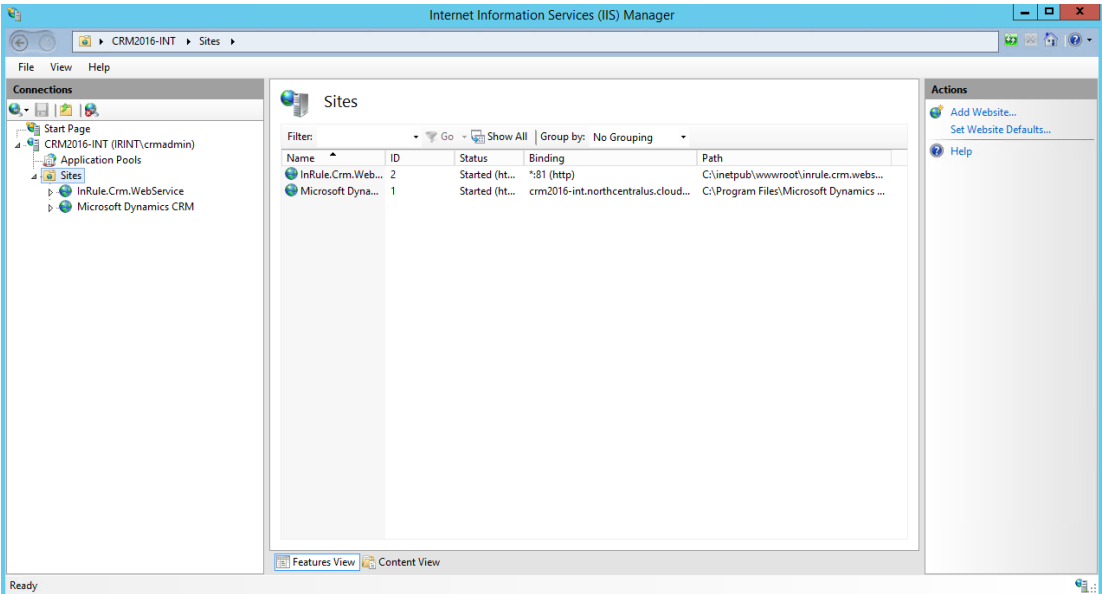
4.3.2 Setting Up the IIS Site for the Rule Execution Web Service

Next, we'll need to setup a website in IIS to host the execution web service. This will require a Windows Server with IIS installed.

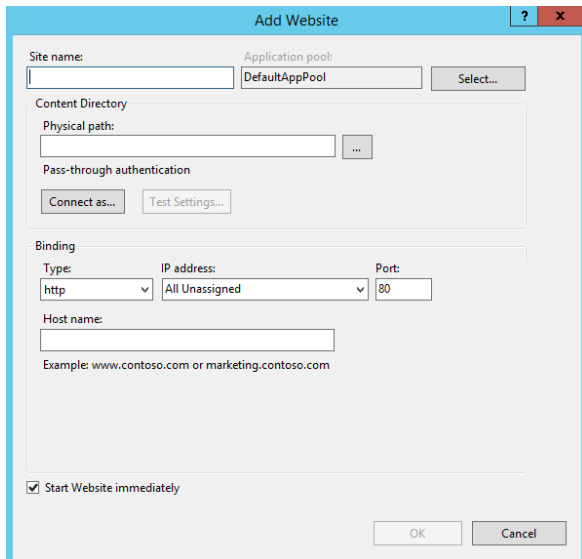
1: Open IIS Manager



2: Click on sites on the left-hand nav pane



3: Click “Add Website” under “Actions” in the right-hand pane



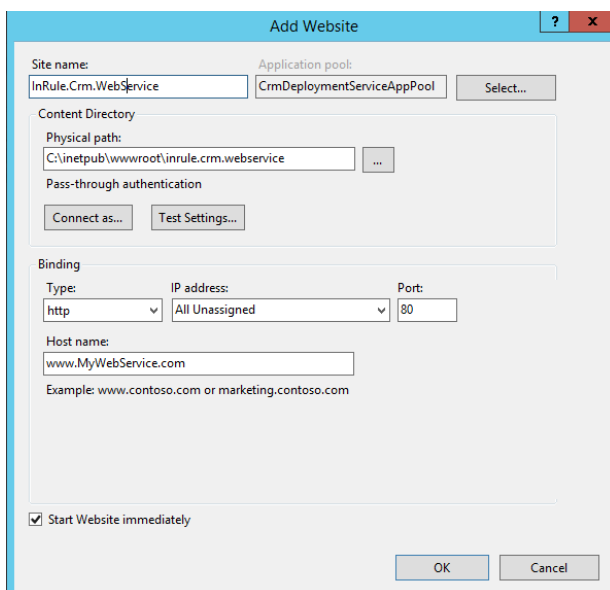
4: Define website information

Define a site name, application pool and host name for the website; these can be configured however makes sense within your organization.

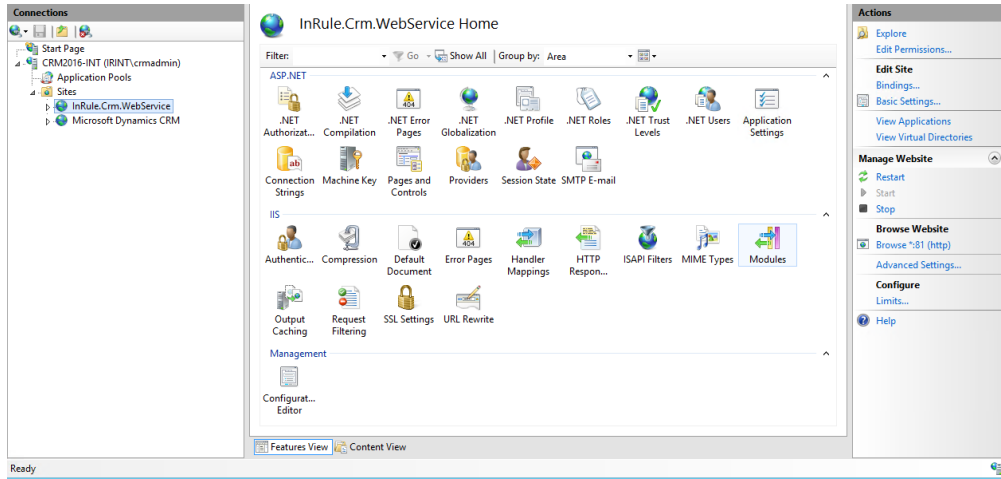
The physical path needs to be set to the actual Dynamics Rule Execution web service folder. This will require you having the InRule components downloaded onto the same server you are setting up the website on

Lastly, select either HTTP or HTTPS, depending on which makes sense for your architecture. If you opt to use HTTPS, you will be required to select a certificate before you can finalize the site.

Once you’ve populated all the required fields, press OK to finalize and start the website



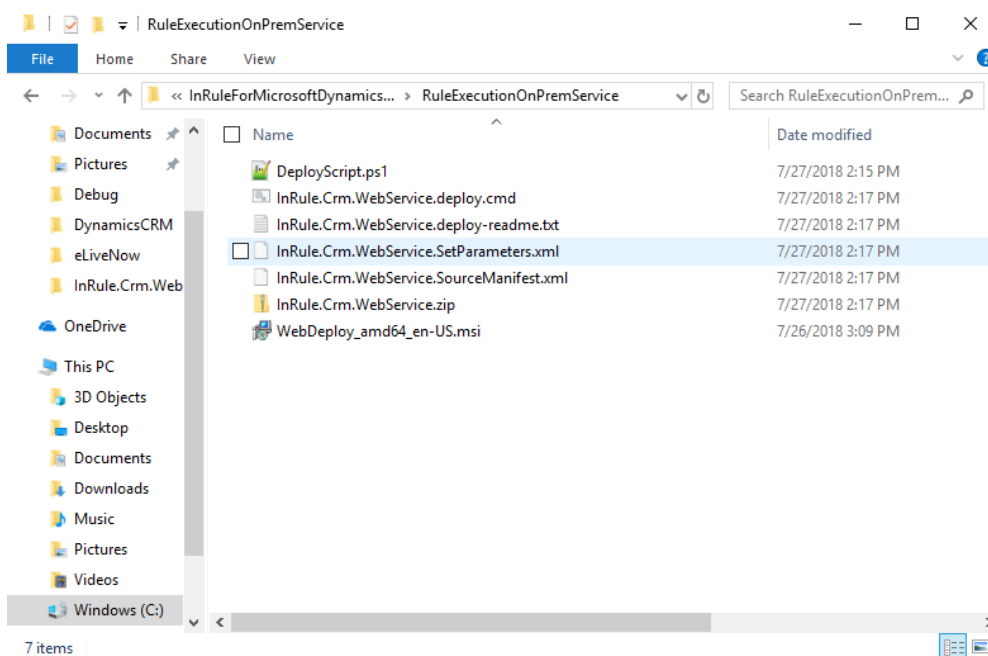
5: Verify your website successfully created



4.3.3 Deploy the Rule Execution Web Service

1: Open your “RuleExecutionOnPremService” folder:

For guidance on how to find this folder, reference the [“Required Files”](#) section above. The folder will need to be copied onto the IIS server you set up your website on in the previous section.




2: Open the InRule.Crm.WebService.SetParameters.xml file in Notepad or a similar text editor

InRule.Crm.WebService.SetParameters.xml - Notepad

File Edit Format View Help

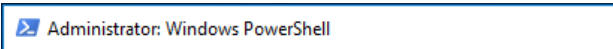
```
<?xml version="1.0" encoding="utf-8"?>
<parameters>
  <setParameter name="IIS Web Application Name" value="InRule.Crm.WebService" />
  <setParameter name="UseInRuleCatalog" value="true" />
  <setParameter name="CatalogUri" value="http://localhost/inrulecatalogservice/service.svc" />
  <setParameter name="CatalogLabel" value="LIVE" />
  <setParameter name="CatalogUser" value="admin" />
  <setParameter name="CatalogPassword" value="password" />
  <setParameter name="CatalogSSO" value="false" />
  <setParameter name="RuleAppDirectory" value="C:\RuleApps" />
  <setParameter name="DynamicsCRM-Web.config Connection String" value="AuthType=AD;Url=; Domain=domain; Username=; Password=" />
</parameters>
```

3: Change placeholder parameter values to real values

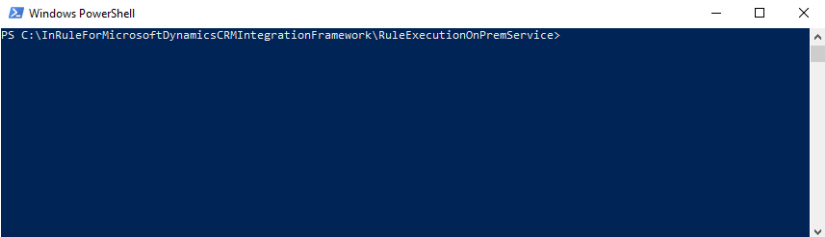
1 IIS Web Application Name	The name of your IIS website as defined in the “Setting up the Catalog Web Service in IIS” section
2 UseInRuleCatalog	Defines if the catalog is being used to store rule apps. Set to true if using the catalog, and false if using file system
3 CatalogUri	The URI for the Catalog Service that will be used
4 CatalogLabel	Defines the label text used to by the service to target the specific rule app in the catalog. Labels are assigned to rule apps in the catalog
5 CatalogUser	Username for Catalog Service, default value is ‘admin’.
6 Catalog Password	Password for Catalog Service, default is ‘password’, please change this using the catalog manager!
7 CatalogSSO	Defines whether to use app pool identity to authenticate to the catalog web service
8 RuleAppDirectory	Specifies the directory where rule apps will be stored if using file system instead of the catalog. If using the catalog, leave as-is.
9 DynamicsCRM-Web.config Connection String	Provide a Dynamics connection string. Information on connection string formatting can be found here: https://docs.microsoft.com/en-us/dynamics365/customerengagement/on-premises/developer/xrm-tooling/use-connection-strings-xrm-tooling-connect  Important: Some customers have reported needing to provide a username in the format “domain\username” in order to successfully connect using IFD.

4: Save your changes

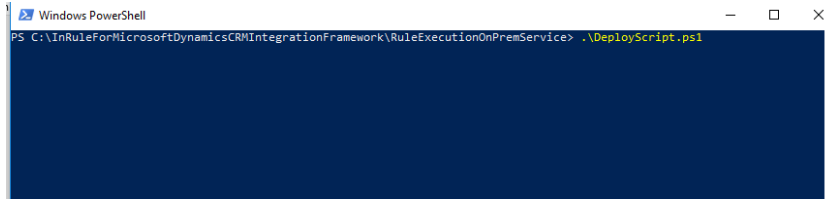
5: Launch PowerShell as an administrator



6: Navigate to the “RuleExecutionOnPremService” folder



7: Run the “DeployScript.ps1” file



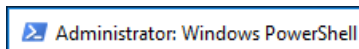
Observe that no errors occur while this script does its work.

It is worth noting that when deploying plugins without isolation in an on-prem environment, Dynamics requires that the user registering the plugin must be added as a Deployment Administrator from Deployment Manager. If the registering user lacks the proper permissions, when deploying the package Dynamics will return an error stating **“Assembly must be registered in isolation.”**

4.3.4 InRule Solution for Dynamics On-Prem Deployment

At this point, all of the InRule server components are setup. The rule execution service should be listening for incoming communication from Dynamics. We must now set up Dynamics. To make this process easier, we will be using PowerShell.

1: Launch PowerShell as an administrator:



2: Navigate to the ‘\Dynamics Deployment’ directory:

```
PS C:\work\CRMIntegrationFrameworkTesting\2.3.4.81\InRuleForMicrosoftDynamicsCRMIntegrationFramework\DynamicsDeployment\tools> cd..
PS C:\work\CRMIntegrationFrameworkTesting\2.3.4.81\InRuleForMicrosoftDynamicsCRMIntegrationFramework\DynamicsDeployment>
```

3: Execute Deploy-CrmPackage.ps1

For Dynamics On-Prem v9, run Deploy-CrmPackage.ps1 with the added argument: -OnPrem.

```
PS C:\> .\Deploy-CrmPackage.ps1 -OnPrem
```

5: Provide Dynamics On-Prem Service Credentials

You will need to login for this script to continue, please be sure to check the ‘Display list of available organizations’ to make sure that you select the correct instance of Dynamics to install to!

Alternatively, the Deploy-CrmPackage.ps1 script also accepts a Dynamics Connection String:

```
.\Deploy-CrmPackage.ps1
```

```
-CrmConnectionString 'AuthType=Office365;Url=https://irroadsandbox.crm.dynamics.com/;Username=CRM_...;Password=...'
```

```
.\Deploy-CrmPackage.ps1 -CrmConnectionString
'AuthType=AD;Url=https://{DYNAMICSURL}.crm.dynamics.com/;Username={DYNAMICSUSERNAME};Password=
{DYNAMICSPASSWORD}'
```

This is a sample connection string; with an on-premises install, there are a wide variety of options that may be relevant, depending on your architecture. For more information on connection string formats, refer to Microsoft's available documentation, found [here](#).

Important: Some customers have reported needing to provide a username in the format "domain\username" in order to successfully connect using IFD.

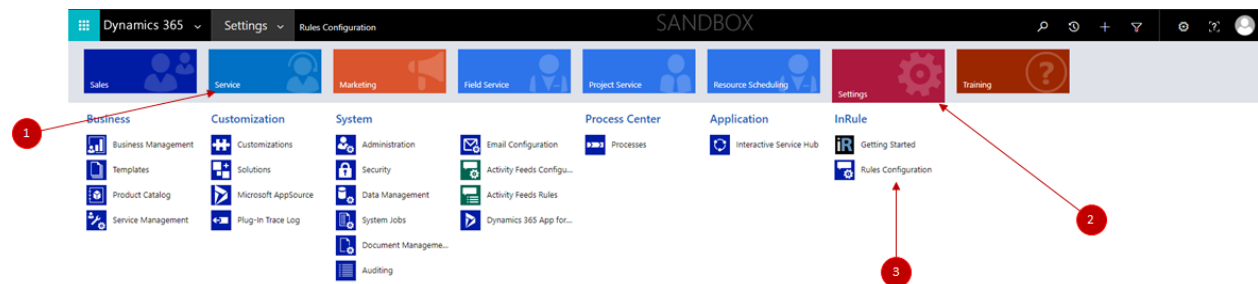
Important: This deployment can take a long time to complete. In most cases it takes around 20 minutes.

Observe that no errors occur while the script is executing.

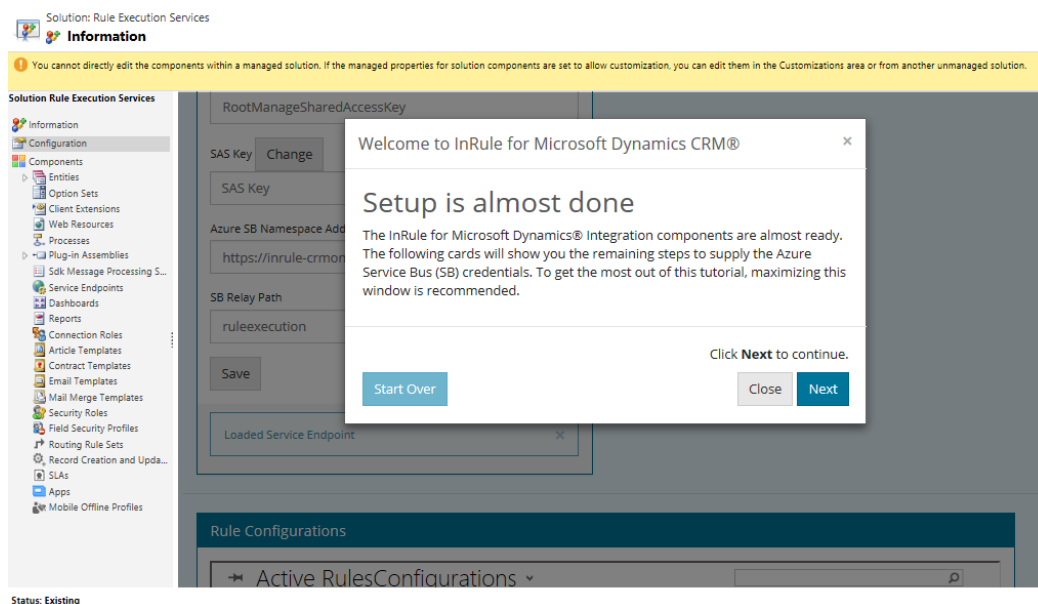
6: Configure the InRule Solution in Dynamics

Next, we need to tell the Dynamics Solution that has been deployed where to find the Rule Execution Service deployed earlier. We will need to have the Rule Execution Service URI ready for this step.

Navigate to the Rules Configuration Page



When the Rule Services Solution is entered for the first time, the Welcome to InRule for Microsoft Dynamics 365 wizard will appear:



Close the wizard, as it is applicable to Online only.

At the top of the InRule Solution configuration page, you will see the Service Endpoint Configuration options for the solution:

Service Endpoint Configuration

Service Endpoint ID

d2e50ca1-ef4c-e611-80e9-6c3be5a82b30

Endpoint Name

Default InRule Service Endpoint

SAS Key Name

RootManageSharedAccessKey

SAS Key

Change

SAS Key

Azure SB Namespace Address

https://inrule-crmonline.servicebus.windows.net/ruleexecution

SB Relay Path

ruleexecution

Save

Test

Do not configure, for Online only

All of these settings can be left as their default values for an on-premises installation, as they pertain to the Online solution only.

Instead of configuring the above, you will need to scroll down all the way to bottom to the “Rule Configurations” section and select the active Rule Configuration

Rule Configurations

Active RulesConfigurations

Search for records

✓	Name ↑	Service Endpoint...	Created On	
✓	Default	http://crm2016-...	7/18/2018 3:39 PM	

1 - 1 of 1 (1 selected)

Page 1

The following popup should appear:

RULE CONFIGURATION - INFORMATION

Default

Modified On: 11/14/2018 9:23 PM | Modified By: First name Last | Status: Active

General

Advanced

Service Endpoint Id (or Uri) * **http://crm2016-int:81/CrmRuleExecution.svc/Execute**

Plugin

Max Plugin Depth: 1
Plugin Retry Interval: 0
Plugin Retry Count: 2
Rule Execution Log Enabled: No

Custom Action Settings

You can configure the naming conventions of this rule configuration as you see fit, but under “Advanced” on the right-hand side of the page, you will need to set the Service Endpoint URI to the URI of your Rule Execution Service endpoint

Advanced

Service Endpoint Id (or Uri) * **http://crm2016-int:81/CrmRuleExecution.svc/execute**

Be sure to click save (the small save button in the bottom-right corner)

Service Endpoint Configuration

Service Endpoint ID: d2e50ca1-e44c-e611-80e9-6c3be5a82b30

Endpoint Name: Default InRule Service Endpoint

SAS Key Name: RootManageSharedAccessKey

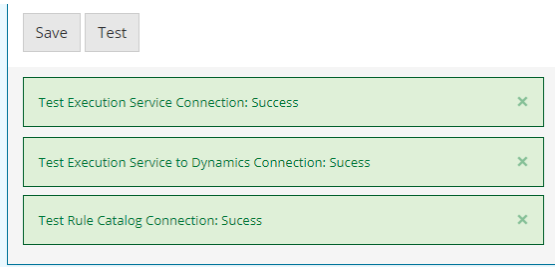
SAS Key: Change

Azure SB Namespace Address: https://inrule-crmonline.servicebus.windows.net/ruleexecution

SB Relay Path: ruleexecution

Save Test

Now that everything has been configured, to can test the configuration by click the test button. This test configuration tests these three things:



1. Execution Service Connection. Test sends a request to the execution service that was deployed and configured to make sure that it is setup correctly.
2. Execution Service to Dynamics Connection. Test makes a request from the execution service to make sure that it can make a request back to Dynamics with the connection information configured on the App Service.
3. Rule Catalog Connection. Test makes a request to the Rule Catalog with the connection information configured on the App Service.

If all the tests succeed, the system has been configured correctly. If any of the tests fail, check the configurations for the subject under test.

7: Verify a successful deployment!

A 'Run Rules' button will now exist on the edit form of all Dynamics entities. The way this button behaves will depend on other settings that have been configured under the Rule Configuration section; this document assumes the default configuration behavior. To understand what settings are available and what they mean, please see [Appendix E: Methods for Executing Rules from Dynamics 365 and Power Platform](#) of this document.

Open up an Account and execute 'Run Rules'. The default deployment is configured to run a Rule App from the catalog named DynamicsRules, which is what we uploaded to the catalog in the Verify Catalog stage. If everything has been set up correctly, you should see the "Rule Execution Completed" message and the description of the account will be updated to provide the date and time.

It is worth noting that the On-Prem plugin will be deployed **outside** of Sandbox mode. To read more on why this is necessary and the resulting implications, reference [Appendix M: Known Issues, Limitations and Troubleshooting](#).

8: Getting Users Ready

While the parties responsible for deploying out InRule can likely leverage the Run Rules button without issue as they're likely to be system administrators, if your organization intends to empower non-admin users to be able to leverage the Run Rules button, they'll encounter permission issues. This is due to rule execution requiring access to various Dynamics configuration entities that normal users don't generally have permissions to touch.

To circumvent this, the InRule Solution for Dynamics ships with a pre-configured role called the "InRule Rule Executor," which contains the bare minimum permission set necessary for users to be able to make use of the Run Rules button. You'll need to assign any users in your organization to this role if the intent is to empower them to run rules through the Dynamics UI.

Appendix A: Additional Resources

Having trouble? Relax! InRule offers many additional resources to help you get InRule correctly integrated with Microsoft Dynamics 365.

InRule's Support Website

InRule's support website can be found at <http://support.inrule.com>. If you do not already have a login for our support site, the client administrator at your company has the ability to create an account for you. If you are unsure of who your client administrator is, please email support@inrule.com.

InRule's Support Team

The support team at InRule is available to help with any product support needs, troubleshooting suspected product bugs, resolving any licensing issues, and free tele-hugs.

The best way to reach Support is through a detailed email sent to support@inrule.com.

You can also reach our support team by calling +1 (312) 648-1800.

InRule's ROAD Team

ROAD Services agreements can be used to engage with ROAD, InRule's professional services team.

ROAD can provide your organization with specialized consulting and tailored Architecture and Authoring Guidance.

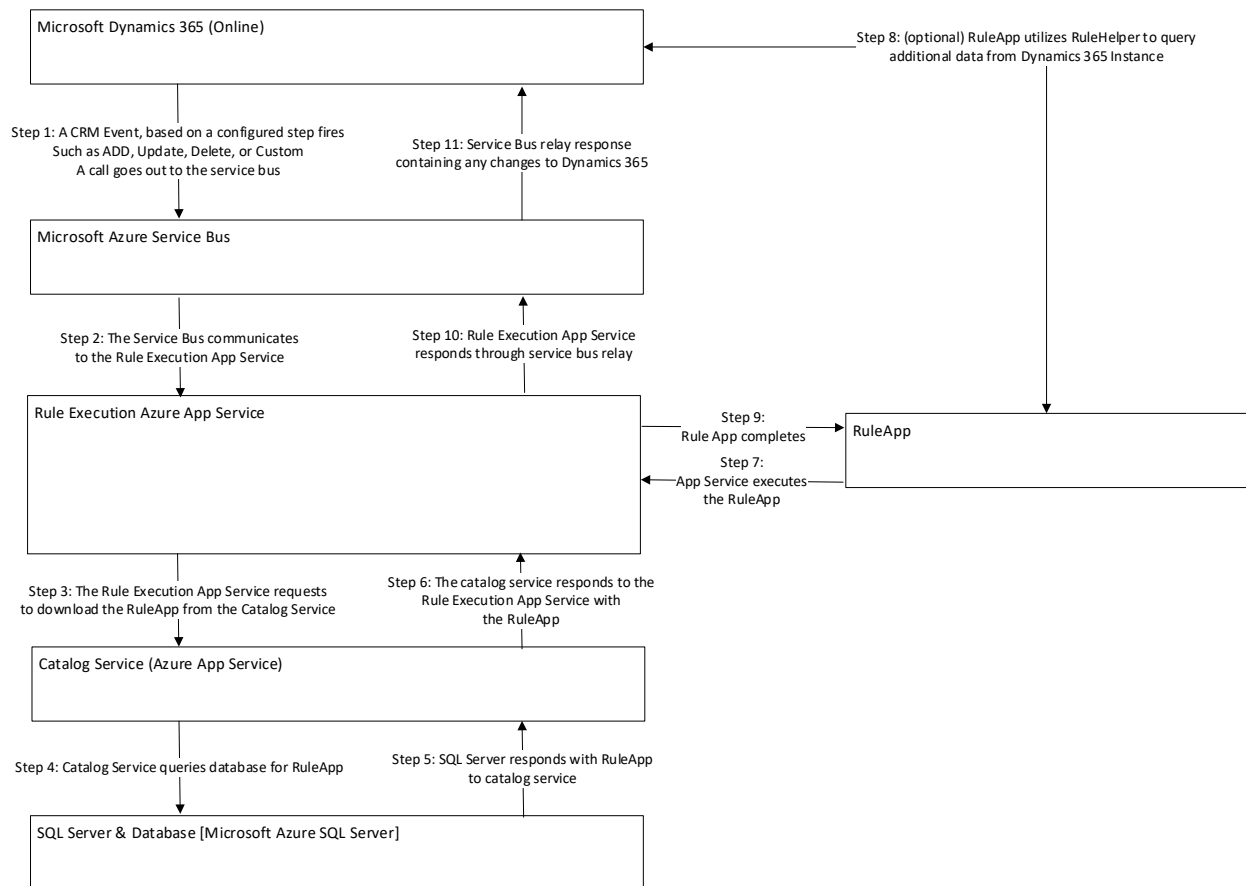
ROAD can assist with less common installation requirements, such as deployment to third party cloud providers or integration with custom software.

ROAD can be contacted by emailing ROADServices@InRule.com

Appendix B: Anatomy of a Request for Execution of Rules Diagram

The below diagram helps to give a top-level understanding of how InRule is integrated with Microsoft Dynamics 365 (Online). Please note this this diagram is a simplification that does not cover topics like caching, iterations, and multiple environments. It serves to show how the request moves through different Azure resources.


1. A Dynamics 365 Event based on a configured step fires, such as ADD, UPDATE, DELETE, or CUSTOM. This generates a call to the Relay, which is setup to relay requests to the connected Rule Execution App Service.
2. The Relay receives the request and relays it to the Rule Execution App Service, which has attached itself to the Relay as a relay listener. For On-Premises, the rule execution service interacts directly with Dynamics.
3. The Rule Execution App Service makes a request to the Catalog Service, asking for a copy of the requested Rule App.
4. The Catalog Service Queries its SQL Server based database for a copy of the requested Rule App.
5. The SQL Server responds with the Rule App.
6. The catalog service responds to the Rule Execution App Service with the Rule App.
7. The Rule App executes inside the Rule Execution App Service.
8. Optionally, the Rule App has an opportunity to query Dynamics 365 for additional data needed to execute rules.
9. The Rule App completed execution
10. The Rule Execution App Service responds through the Azure Relay
11. The Azure Relay relays the response to Dynamics 365, where the receiving plugin synchronizes changes.



Appendix C: General Authoring and Integration Concepts

Runtime Mapping across Nested Relationships

Much like Dynamics 365, the InRule rule engine offers strong support for hierarchical and relational data. Within a given Rule Application, data can be considered across parent-child relationships within a single rule request. These relationships can take the form of Collections (1 - * relationships) or 1 – 1 relationships.

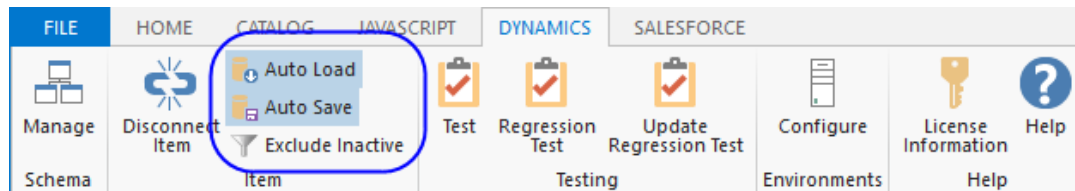
 **Note:** When N:N relationships are imported into InRule, they behave as 1:N Collection relationships within the Rule Application.


In addition to the abilities of both products to handle relational data, both products also offer the ability to declaratively configure “Entities” and “Fields”. Both products also allow for different strongly-typed Entities and Fields to be accessed with loosely-typed SDK interfaces. Because of these inherent similarities and flexible interfaces, it is possible to build a reusable mapping component that can convert any given graph of loosely typed Dynamics Entities to InRule Entities, and vice versa.


Controlling irVerify Behavior with Load, Save and Inactive Record Settings

When working with a tree composed of many related Dynamics Entities, it is often useful to have explicit control over which relationships are either automatically loaded or automatically considered in change detection for persistence. If a relationship is skipped during the initial load routines, then it is available to be conditionally populated later using rules.

In the irX rule authoring ribbon, there are three buttons that give the rule author the control to denote if a relationship should be automatically loaded, saved or have inactive records excluded.



 **Note:** Automatic loading and saving is enabled by default for all relationships that are imported from Dynamics. The rule author can opt-out of these automatic behaviors by unselecting “Auto Load” or “Auto Save”. When these buttons are selected, metadata attributes are written into the Rule Application for the given relationship. These metadata attributes are used by the irVerify data loader when recursively loading data or detecting changes for persistence.

 **Important:** If loading or saving is disabled for a given relationship, then it is also disabled for all Entities that are children of that relationship.

Since Microsoft Dynamics 365 uses the “inactive” status to do a soft delete of records, it is possible to include both active and inactive records when loading a Collection or relationship. You can use the “Exclude Inactive” button to exclude inactive records, returning only active records. To ensure that this feature works correctly, the child Entity of the Collection or relationship should map the Dynamics 365 **Status** field.

Polymorphic Lookups

Usually, when setting up an entity relationship in Dynamics, the relationship can only apply to one entity type. But, polymorphic lookups can refer to any of the entity types that are set up for the relationship. For instance, a 'Payment' entity could be linked to many different kinds of custom entities that represent items that can be bought in a company.

Just like Dynamics, InRule entity relationships usually point to only one type of entity. When importing polymorphic lookups in the irX schema manager, a special linking entity is created automatically in the rule app. This linking entity has a type field that shows which entity type is being used, and one linking field for each possible entity type in the relationship. These values will be populated at runtime and allow writing rules to say if Entity Type = Course, then... or if Entity Type = Book, then...

In the below example, the **AffectedBy** field on Contact is a polymorphic field that can be of type **Opportunity** or **Quote**. When importing the AffectedBy field, it will generate an associated entity called **ContactAffectedBy** that is used as a container to represent the variable relationship to Opportunity or Quote. This pattern will be used for each polymorphic entity imported in the schema and will allow for the reading and setting of associated entity references.

The screenshot displays the InRule schema manager interface. On the left, the 'Entities' pane shows a tree structure of entities: Account, Contact (expanded), CustomerLookup, Opportunity, Quote, and ContactAffectedByLookup. Under 'Contact', the 'AffectedBy' field is highlighted. On the right, the 'Field' configuration pane for 'AffectedBy' is shown. It includes fields for 'Display name' (Affected By), 'Data type' (Entity), and 'Entity' (ContactAffectedByLookup). There are checkboxes for 'Parent context' and 'Default value'. A 'Value list' dropdown is set to '(None)'. Below the 'Field' section, there are sections for 'Metrics' (Output as metric), 'Vocabulary' (with a link to 'Create vocabulary'), and 'Rules'.

Appendix D: Accessing Dynamics 365 Directly from Rule Helper

In the default Rule Execution setup, all relationships between Entity types must be established before these entities can be used by the rule app. This behavior is intuitive, but it is not ideal for all business problems. The InRule Integration framework provides a 'Rule Helper' assembly that can be used directly in a rule app and allows rules to load, compare, and assign data that is not related in Dynamics before rules are executed. The [Integrating the Rule Helper Component](#) section of this appendix provides more information for setting up a Rule App to use the Rule Helper.

When to use the Query from Rules Approach

The query from rules approach adds value for the following business problems:

- The rules need to reference “lookup” information that may be in a list or set of Entities that are not specifically related to the current Entity hierarchy
- The purpose of rules is to create new relationships between Entity instances that already exist in Dynamics
- The rules need to compare many combinations of unrelated Dynamics Entities and produce results about best possible matches or scores
- A custom filter is required when loading data for 1:N or N:N relationships
- The rules need to delete an entity from Dynamics instead of only removing the relationship



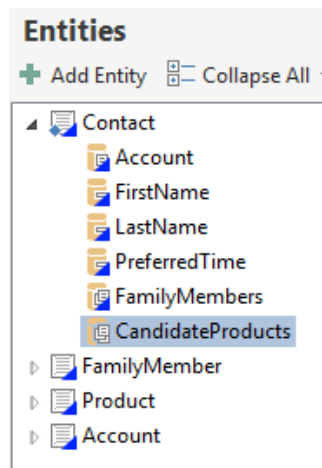
Important: Deleting an entity with relationships configured to cascade delete will also delete those related entities. Carefully review your entity schema and relationships before using the delete functionality in the Rule Helper

Working with Disconnected Fields when Loading and Saving Data

One of the most important integration concepts when loading Dynamics data from rules is the notion of “Disconnected” Fields and Fields that have “Auto Load” and “Auto Save” disabled.

irX allows the rule author to explicitly control the “Auto Load” and “Auto Save” behaviors of Fields that are connected to Dynamics.

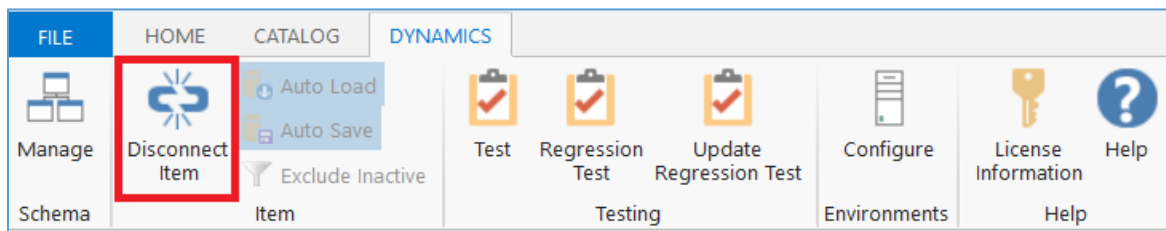
The example below shows a Collection named CandidateProducts. Since the Collection is not marked with a blue triangle, it is not considered to be attached to Dynamics.



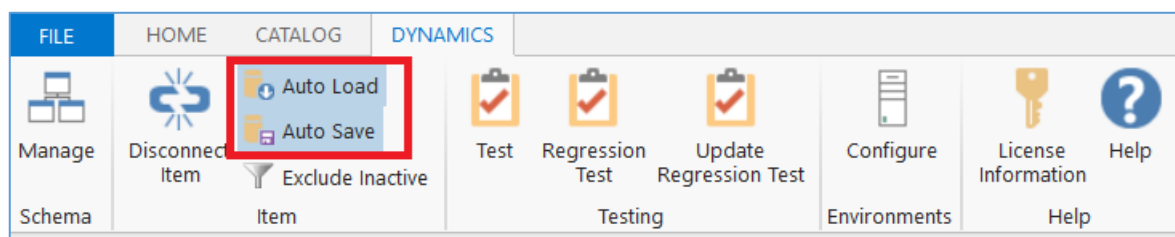
Important: Although Entity Fields and Collections may be “Disconnected” from Dynamics, the types contained by the Collections can be set to types that were imported from Dynamics.



Note: If a Field is added to the schema using irAuthor, then it will be disconnected from Dynamics. If a Field has been imported from Dynamics using irX, then it can be disconnected from Dynamics by clicking the “Disconnect Item” button in the irX ribbon.



Important: Two additional settings appear in the irX ribbon that offer additional control over automatic loading and saving behaviors for Fields that remain connected to Dynamics. In the example below, both buttons are “lit up”, which denotes that the settings are enabled. By default, automatic loading and saving is enabled for all Fields that are connected to Dynamics.

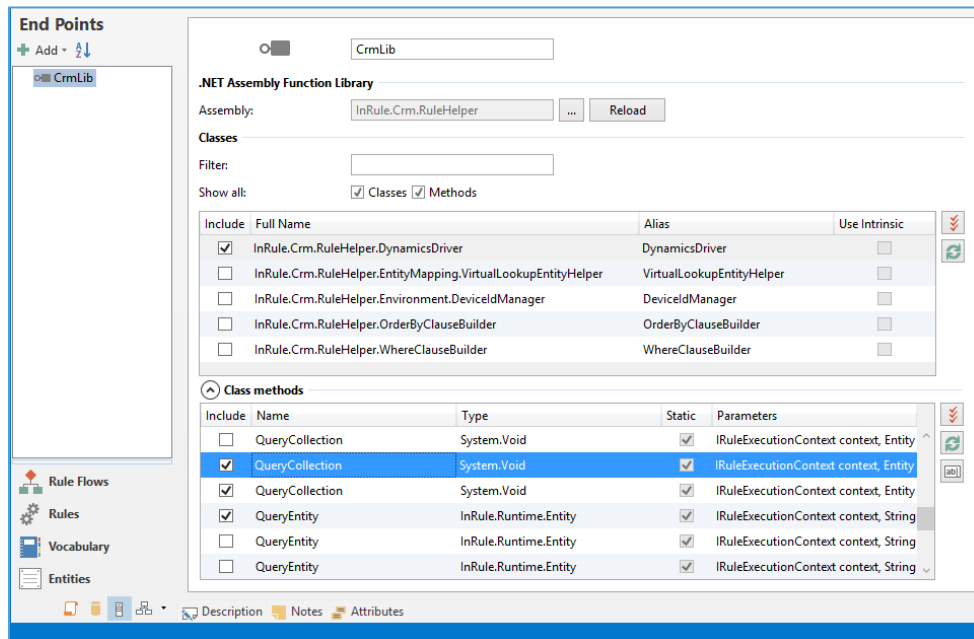


Integrating the Rule Helper Component

InRule provides [a sample rule application](#) (DynamicsRules) that is already configured for RuleHelper usage. You can simply edit this rule app, or, if you wish to integrate RuleHelper into an existing rule app, you can copy both the UDF Library “RuleHelper” and End Point ‘CrmHelper” from the DynamicsRules rule to another rule application.

If you wish to manually create the UDF Library and End Point in irAuthor, follow the steps below.

1. Create a new rule app using the irX add-in for irAuthor.
2. Create a new “.NET Assembly Function Library” end point and bind the end point to the InRule.Crm.RuleHelper.dll assembly. Select the DynamicsDriver class and then select the methods that should be callable from rules. Edit the name of the end point to “CrmLib” or similar. Select the methods from the DynamicsDriver that are needed for the Rule Application. You do not need to select all the methods—only import the methods that will actually get used by rules. Additional methods can always be imported later by revisiting the endpoint screen and reloading the assembly.



3. Add a User Defined Function library and set the name to “CrmHelpers” or similar. This library will contain functions that the rules will call to query Dynamics.
4. Add a User Defined Function to the new library. The example below shows a UDF that will be used to execute the QueryCollection method on the DynamicsDriver. Fill out the UDF with script that will call a method on the DynamicsDriver.

Note: The methods on the DynamicsDriver are designed to be reused for more than one Entity type, Field, or set of Fields. The name of the Target Field or Collection should be supplied as a string. When querying a Collection of results, an optional “where” clause can be provided that will be forwarded to calls against the CRM SDK. In addition, an “order by” clause can be provided to return sorted results.

Important: This integration pattern relies on the “Context” object that is available from irScript. The Context object returns information based on the context under which a given UDF is executed. For example, when executing an Entity Rule Set, the Context.Entity returns a reference to the Entity against which the current Rule Set is executing. The Context and its child properties are passed to the DynamicsDriver so it has enough information to form calls to Dynamics 365 and map responses back to the InRule Rule Session.


Important: When using Context with a Decision, Context.Entity cannot be used by Decisions. Instead, the UDF will need to pass the entity in as a parameter.

Note: The Context.FunctionLibraries property can be used to create calls to the .NET assembly library methods, such as the methods imported in Step 5 above. The following script example demonstrates how to use the Context object in irScript to form a call to a static .NET method:

```
Context.FunctionLibraries.DynamicsDriver.QueryCollection(Context,
Context.Entity, collectionName, filter, orderBy, connectionString);
```

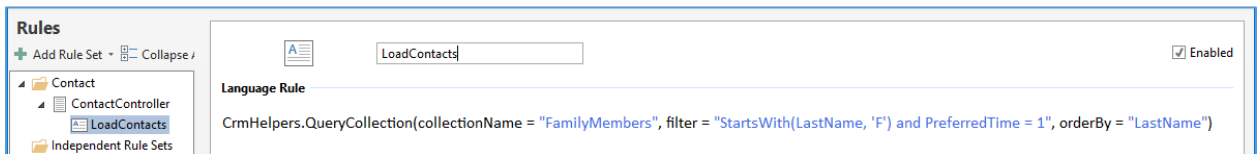
Important: The “connectionString” argument is optional due to the fact that the overloading of the methods on the DynamicsDriver is not passed in the by the rule engine or irX. It will then be looked up from either the .NET config file based on the given environment. As a rule


of thumb, it is generally not recommended to override the connection string in this manner as it can perpetuate difficult debugging scenarios and inadvertent dependencies.


 **Important:** Below is a sample connection string that needs to be defined in the irAuthor.exe.config XML for native irVerify to call the RuleHelper. For information on Dynamics connection string formatting, refer to <https://docs.microsoft.com/en-us/dynamics365/customer-engagement/developer/xrm-tooling/use-connection-strings-xrm-tooling-connect>.


```
<connectionStrings>
  <add name="DynamicsCRM"
    connectionString="AuthType=OAuth;Username=jsmith@contoso.onmicrosoft.com;
    Password=passcode;Url=https://contoso.crm.dynamics.com AppId=51f81489-12ee-4a9e-aaae-
    a2591f45987d;RedirectUri=app://58145B91-0C36-4500-8554-080854F2AC97;LoginPrompt=Auto "/>
</connectionStrings>
```

5. Rules can now be authored to execute methods on the DynamicsDriver. These methods can be used to load Collections, single Entities, or single Fields from Dynamics based on conditional logic within rules.



 **Note:** The example above includes a call to the default business language templates for a method on the DynamicsDriver. The InRule vocabulary features can be used to modify these templates to be more user-friendly for business users.


 **Important:** The Target Collection in the sample rule is called "FamilyMembers". This is a Field that either does not exist in Dynamics (only for use in rules), or has been imported and then "disconnected" from Dynamics using the "Disconnect Field" button, or has "Auto Load" disabled.

 **Note:** Please see the following sections for more details on the creating the "filter" clauses similar to the one used this example.

Filtering Queries using the Where Clause Builder

When loading data from Dynamics during rule execution, it is critical that the rule author is able to author logic to specify which Entity data to load. Using the RuleHelper, this is accomplished by allowing the rule author to pass in a "filter" or "where" clause into the calls against the DynamicsDriver class.

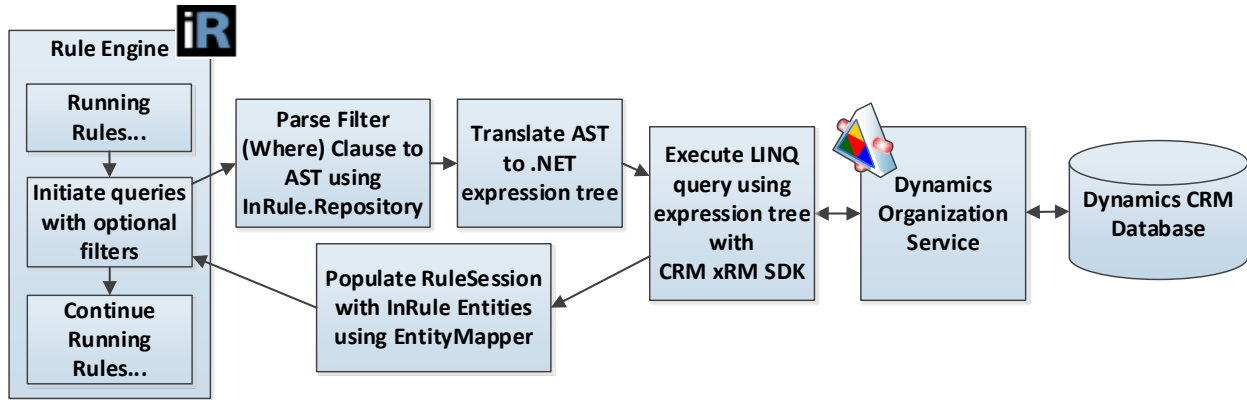
During execution of the DynamicsDriver, the filter clause is parsed into an Abstract Syntax Tree (AST) and then translated into a LINQ expression tree, so it can be consumed using the CRM SDK. The filter clause is based on the InRule function syntax format.

 **Note:** The InRule function syntax format is used for the following reasons:

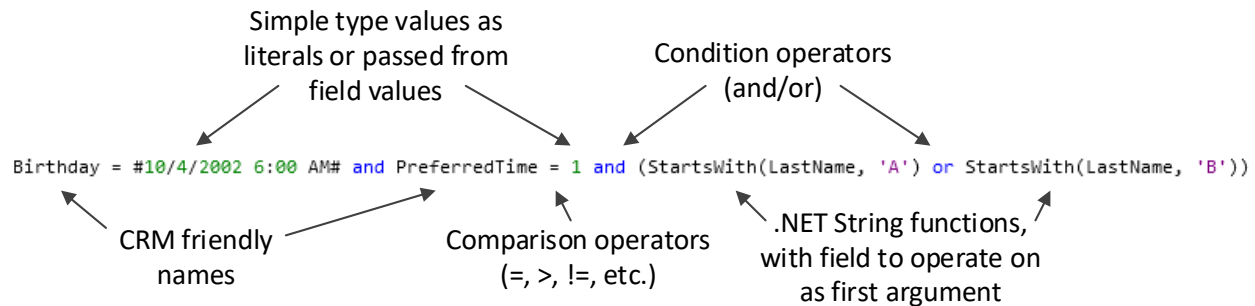
- The syntax rule format is consistent with the rule authoring experience used throughout irAuthor

- This format can make good use of the InRule AST parser that is included as part of irSDK

The diagram below depicts the logical flow of steps used by the DynamicsDriver and WhereClause builder classes to query data from rules.



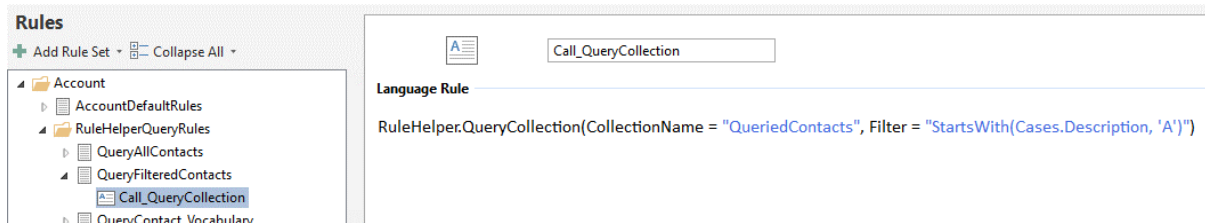
The diagram and notes below contain some additional information about forming the filter clause in a rule:



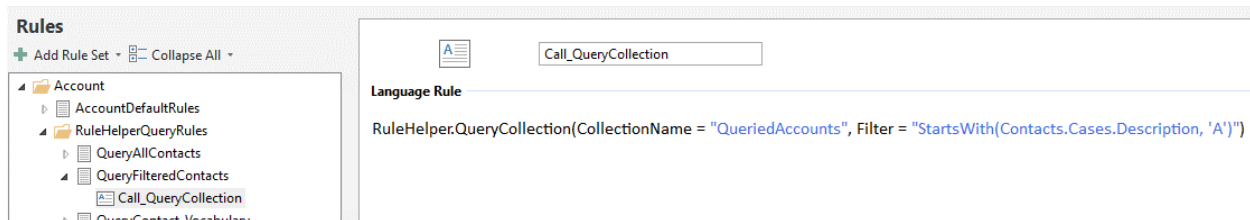
- All the Field names that are used are the Dynamics friendly names that are used in the Rule Application. These names are mapped back to the Dynamics Field names in the parser.
- String literals should be wrapped in single quotes, date literals should be wrapped in pound signs.
- Simple operators are supported to compare values, such as `=`, `!=`, `>`, `<` (ex. `Age > 21`, `Name != 'Ralph'`).
- Multiple conditions can be chained together using 'and' and 'or' keywords.
- The expression tree builder will attempt to call a given .NET framework function if it appears in the expression.
 - The .NET call must be expressed as a function, with the first argument being the name of the Field to operate on. The remainder of the arguments will be passed through in order to the .NET function call when it is formed.
 - Examples: `StartsWith(FirstName, 'A')` `EndsWith(LastName, 'ez')` `Contains(LastName, 'Smith')`
 - Any non-static methods on the .NET String class are candidates to be called as functions using the parser syntax above. However, InRule has only tested the `StartsWith`, `EndsWith`, and `Contains` methods on the String class.

- The following keywords and operators are supported by the InRule AST parser and expression tree translation code: =, <>, !=, +, -, *, /, or, and, xor, >, >=, <, <=, ^, %

The filter expression also supports querying against related entities, simply by appending the related entity name in front of the relevant query field. Querying against related entities requires that all **entities** queried in the relationship chain be imported into irAuthor. The queried **fields** on each of the queried entities do not necessarily need to be imported, but if they are not, the entire Dynamics schema field name must be used, rather than their irAuthor aliases.

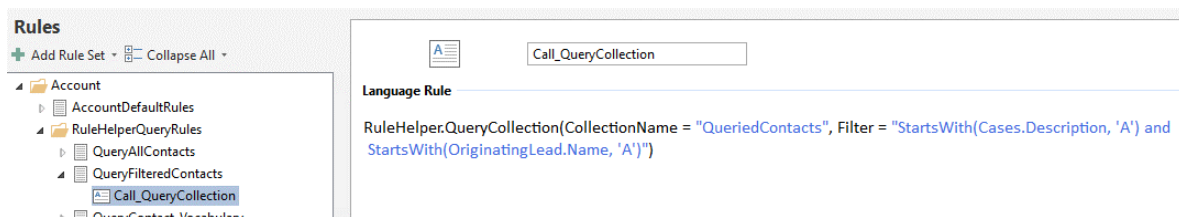


In this example, we are populating a collection by querying the Contact entity, which is the “parent” entity here. We are then applying a filter statement to return only contacts with related Cases that have Descriptions starting with “A.” Cases in this example is the “child” entity. Notice how the hierarchy of the related entity down to field is denoted. If you wanted to drill down another layer to a “grandchild” entity (in this example, an entity related to Case), it would be accomplished by simply continuing the chain from entity to field. Below is an example of a “grandchild” case:



This example would return Accounts that have related Contacts with Cases with Descriptions starting with “A.” The filter expression can support querying in this manner up to 15 “layers” deep, including the initially queried entity. Put another way, you can have up to 15 different related entities in a single filter expression.

The filter expression supports querying against multiple properties from different related entities. In the below example, we are querying for Contacts with Cases that have Descriptions starting with “A” and also have Leads with Names starting with “A.”




Ordering Query Results with the OrderByClauseBuilder

The DynamicsDriver class also supports the ability to control the order of the results returned from Dynamics by passing in an optional “order by” clause. The order by clause can accept only a single Field

name, which should be the name of the Field in the Rule Application. The results are always sorted in ascending order, unless the Field name is followed by the “desc” syntax. Please see the examples below:


To sort ascending, pass the Field name to use in the sort:



Language Rule


```
CrmHelpers.QueryCollection(collectionName = "FamilyMembers", filter = "StartsWith(LastName, 'F') and PreferredTime = 1", orderBy = "LastName")
```


To sort descending, pass the Field name to use in the sort followed by the “desc” keyword:



Language Rule

```
CrmHelpers.QueryCollection(collectionName = "FamilyMembers", filter = "StartsWith(LastName, 'F') and PreferredTime = 1", orderBy = "LastName desc")
```

 **Note:** The “order by” clauses generally contain much simpler expressions than “where” clauses. However, InRule syntax rules format is used for the order by clause to be consistent with the where clause approach.

 **Important:** The parsing and translation of the order by clause is handled in the Rule Helper contained in the

Methods Available in the Rule Helper

The following table lists the public, static methods that are available in the DynamicsDriver

Method Name	Description
LoadMappedChildCollection	Populates a child Entity Collection based on an existing 1:N relationship in Dynamics. The Collection is populated based on existing parent-child relationship data in Dynamics.
LoadMappedChildEntity	Populates a child Entity Field based on an existing 1:1 relationship in Dynamics. The Field is populated based on existing parent-child relationship data in Dynamics.
QueryCollection	Populates an Entity Collection with a set of a given Entity type. An optional filter clause (where clause) can be used to define selection criteria for the Entity set. The Collection does not need to correspond to a 1:N relationship in Dynamics.
QueryEntity	Populates an Entity Field or variable based on a query to Dynamics. An optional filter clause (where clause) can be used to define selection criteria for the Entity. The Field does not need to correspond to a 1:1 relationship in Dynamics. If more than one Entity is returned from the query to Dynamics, then the first Entity in the set is used.
QueryField	Populates a primitive Field or variable based on a query to Dynamics. An optional filter clause (where clause) can be used to define selection criteria for the Entity. If more than one Entity is returned from the query to Dynamics, then the Field value from the first Entity in the matching set is used.
QueryNtoNCollection	Loads entities across an N:N Collection that has been defined in Dynamics. The relationship name and parent and child ID Fields must be provided.

Method Name	Description
LoadMappedNtoNCollection	Similar to QueryNtoNCollection, except that the relationship must be imported into the Rule Application.
DeleteEntity	Deletes an entity referenced by a lookup field. Unlike other rule helper methods, the deletion will not be performed immediately, but will be marked for deletion and then processed with other changes returned to Dynamics from the execution service
DeleteEntityFromCollection	Similar to DeleteEntity, but takes a reference to a collection and index instead of a lookup field.
GetUserLocalTimeFromUtc	Converts a Dynamics UTC time value into the local time for a given user. This method makes use of the LocalTimeFromUtcTimeRequest that is part of the CRM SDK.

Additional Flags Available to Control Loading and Caching Behaviors in the Rule Helper

During a given query operation, there may be advanced use cases that require specific control over loading or reloading data from Dynamics 365. The optional overloads of the QueryEntity and QueryCollection methods expose a set of optional Boolean flags that help control caching and depth of loading behaviors. The table below list these parameters:

Parameter Name	Description
loadChildren	Denotes if the execution service should recurse the Entity graph and load all children. If false, no children are loaded below the Collection Members that are loaded. The default value is true.
useCaching	Denotes if previously loaded Dynamics Entities should be reused from the InRule entity cache, or if new entity instances should be created. If false, the original entity data will be requested from Dynamics, and a copy of the Entity is created. The Instance ID is not set to the GUID of the Dynamics Entity, which will also prevent changes to this entity from being written back to Dynamics. This functionality can, for example, be used to load the original values for an entity persisted in Dynamics when rules are run on update and compare the original and updated values. The default value is true.
overwriteIfLoaded	Denotes if a previously loaded Dynamics Entity should be repopulated with the latest values in Dynamics. This behavior will overwrite Field values stored in the cache. The default value is false.
cacheInAppDomain	Denotes if the result of the query should be saved in the persistent AppDomain cache. The difference between this parameter and the 'useCaching' parameter above is that enabling this parameter will save the query result in a cache that will persist across multiple different rule executions, where the above parameter only enables caching within the scope of a single rule execution. For more information, refer to Configuring the AppDomain Cache

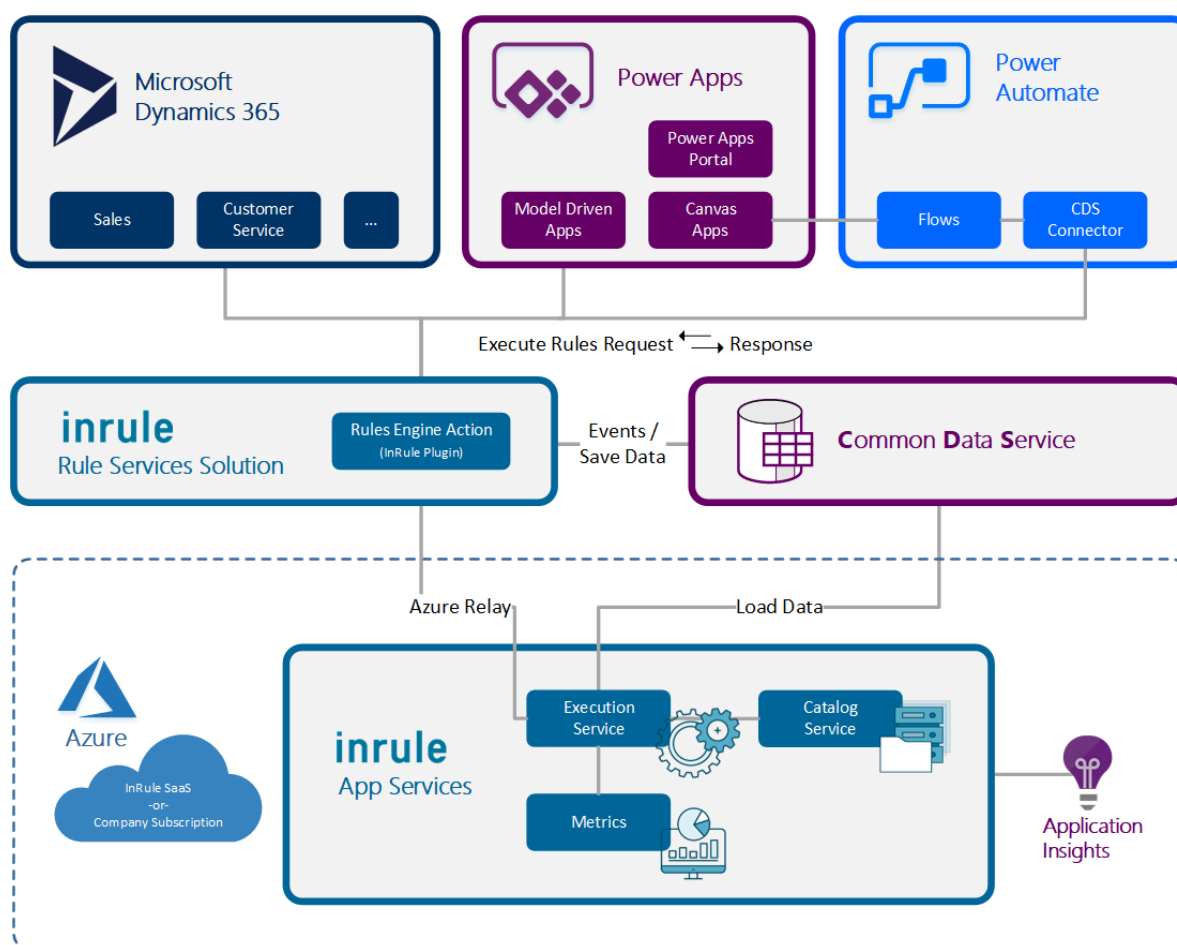


Note: The default values should always be used for the cache settings unless there is a specific use case that requires different behaviors.

Appendix E: Methods for Executing Rules from Dynamics 365 and Power Platform

InRule supports several methods for the execution of rules from both Dynamics 365 and Power Platform. As Microsoft has evolved its singular app platform vision across Dynamics 365, Power Apps, and Power Automate, many of the common integration techniques employed for one or the other, can now be extended to all. This is made possible by Dataverse which is the backbone data integration service across Dynamics 365 and Power Platform. InRule for Dynamics 365 is designed to work directly with Dataverse entities for event handling, data loading/saving, and schema mapping functions.

The diagram below depicts the high-level integration flow for InRule with Dynamics 365, Power Apps, and Power Automate in an Azure Cloud Environment.



The following sections provide a detailed overview of the methods for executing rules and how they apply to various aspects of Dynamics 365 and Power Platform. While the listed methods provide proven coverage across most functional needs, your implementation needs may vary, so this guide provides more technical details to help you create the right solution.

Method of Rule Execution	Dynamics 365	Power Platform			
		Power Apps			Power Automate
		Model-Driven Apps	Canvas Apps	Power App Portal	
Dataverse Events (Plugin events)	●	●	●	●	●
Rules Engine Action (InRule Plugin)	●	●	◐	◐	●
Run Rules Button	●	●			
Workflow Activity	●	●			
Form Events	●	●			
JavaScript	●	●			

One thing that is consistent across all options is that each support mapping entity data to be used in execution of rules. InRule for Dynamics supports both explicit and auto rule sets, in addition to decisions. All these options can be used together, and each have factors to consider when choosing which to use. Some of the key considerations are outlined below.

User Control

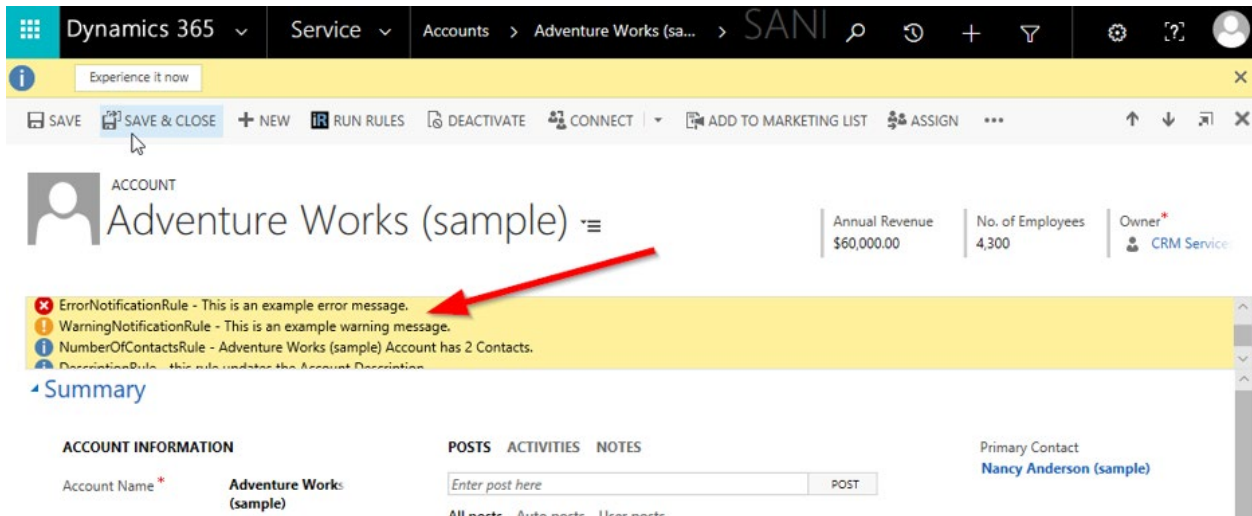
If users need to be able to execute rules on demand, the Run Rules button and Workflow activity are the easiest choice to work with. The Run Rules button is displayed on the command bar of every entity, so it can easily be clicked at any time by end users to get feedback from rules. Using the custom workflow activity also allows rules to be run on demand, potentially as part of a more complex process that contain a rules processing step.

Rules can be invoked indirectly by the user by tying rule execution to events within Dynamics. The typical way to accomplish this is by configuring the InRule plugin to run on entity Create or Update events. Form events like field change or form save can also be used to trigger rule execution. For more advanced form scenarios, like integrating with another command bar button, custom JavaScript can also be used. The included Custom Action can even be used to respond to user events outside Dynamics, by providing a web API endpoint that can be accessed by external systems.

Display in the User Interface

If you want to display information to the end user based on rule execution, the run rules button is the easiest method. When executing rules from the button, any Errors, Warnings and Informational notifications will be shown in the notification pane, along with any Validations. Executing rules via the

included form events helper function or calling the `executeRules` function from custom JavaScript will also display the same information.



Using the custom workflow activity itself will not display any data in the UI, but the workflow activity does provide output variables with the results of rule execution that can be displayed in a dialog or output via some other method if desired.

Plugin events will not typically display any info in the UI, but any errors or validations will show up in the plugin error pop-up if the plugin is set to run synchronously.

Execution Against Dirty Data

If you need to execute rules against dirty form data, or any entity state other than what is saved to the database, you can enable the 'Use Dirty Entity Image' setting as described in [An Explanation of the InRule Custom Action Options](#). If this option is enabled when running rules from the 'Run Rules' button, the current form values will be collected and sent to the rules. This can be used to perform form value validation, or to provide insight into a what-if scenario prior to changes being saved. This same config value also governs the behavior for form events or any custom JavaScript that calls `executeRules`.

The custom workflow activity and plugin events do not provide the ability to specify a dirty entity image. They can only run against the data provided by the plugin pipeline. However, in the case of a plugin entity update event the entity change image is automatically included. In addition, the custom action provides a `dirtyEntityImage` field that accepts a json-serialized entity image.

Responding to API Events

If you want to always execute rules when an entity changes, no matter how those events are initiated, you need to register the included plugin to an event. Running rules based on plugin events means that not only will rules be executing when creating or saving entities from within the Dynamics 365 Entity Forms UI, but they are also executed when entities are changed through API calls.

API calls are frequently consumed by 3rd party add-on user interfaces, Extract-Transform-Load (ETL) synchronization with other software products, and by custom software solutions as an integration point with Dynamics 365. By utilizing this option, an implementer can take advantage of rule execution for needs beyond the Dynamics 365 Entity Forms UI.

1 Dataverse Events

The InRule Solution contains a plugin that can act as a handler for events fired by Dataverse. By registering new steps to Create or Update events against Dataverse Entities, rules execute against the entities that are associated with an event as it occurs.

Configuring Plugin Events

You can register plugin steps using the InRule Rules Configuration page included with the InRule Solution. Alternatively, for advanced scenarios, you may use the Plugin Registration Tool from the Microsoft Dynamics 365 Software Development Kit (SDK), but this is typically only needed for special circumstances where greater plugin management is warranted.

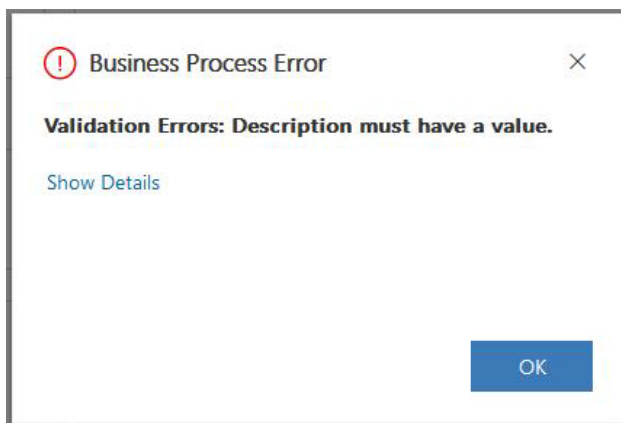
There are two steps required to register a plugin event:

1. **Create a Rule Configuration Record** (or use the Default Rule Configuration) - create or update a Rule Configuration record according to the steps in [Updating InRule Rules Configuration Records](#)
2. **Associate the configuration record** to the create or update message of an entity by following the steps in [Associating an InRule Configuration record to an Entity](#)

Pre-Operation vs Post-Operation for plugin step registration. Plugin step registrations can be configured to run on either Pre-Operation or Post-Operation. By default, the InRule configuration page will register all plugin steps to Post-Operation. In some scenarios Pre-Operation may be preferred. For example, if there are both create and update plugin steps registered to the same entity, using Post-Operation for create can result in extraneous update messages being fired when running rules on create. The create plugin step registration can be changed to Pre-Operation using the Microsoft Plugin Registration Tool. There are a few limitations to Pre-Operation plugin step registrations. Since Pre-Operation takes place before the entity is saved to Dataverse, new entity associations and status code checking will not work.

Validation and Cancellation

When running rules from a plugin, you may want to cancel any changes that have been made to the entity as well as any changes made from rules. One way to accomplish this is by returning a validation error from the rules. If the plugin detects any validation errors, it will skip saving any changes made by the rules and stop the plugin pipeline to prevent any further changes. If an end user is saving or creating an entity and this happens, they will see the following error:

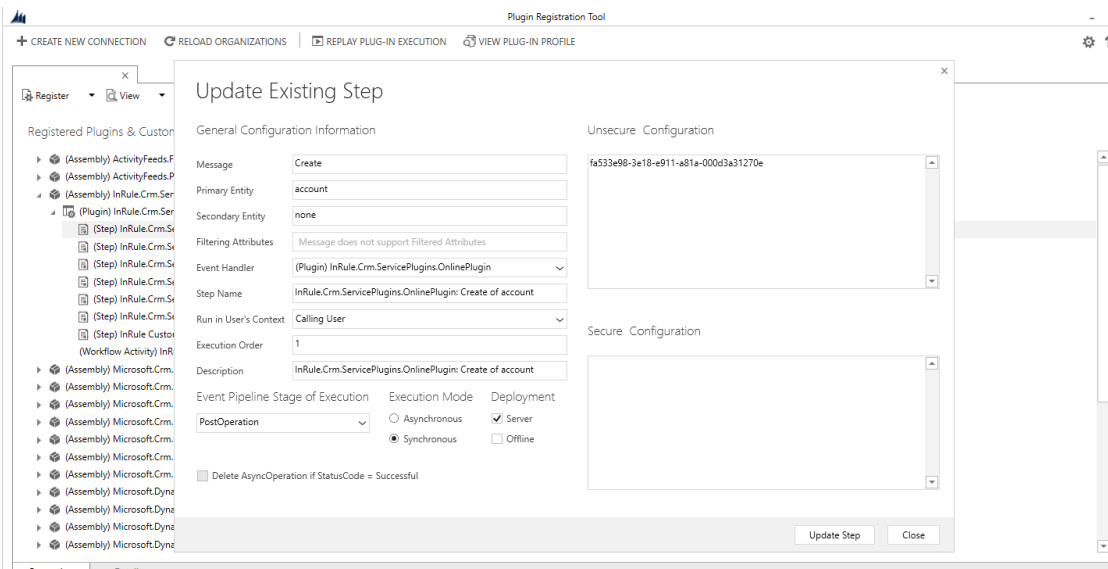


For a better validation user experience, you can also run rules on the 'OnSave' event of a form, as described in the [Form Events](#) section.

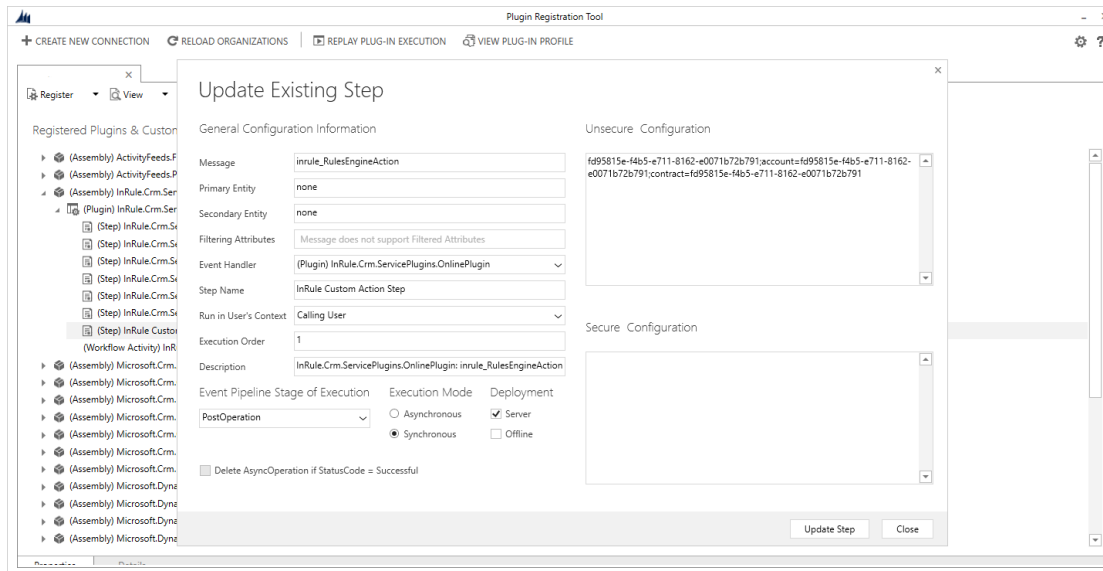
Plugin Registration Tool (Advanced Scenarios Only)

Important: This tool registers all steps to Post Operation stage of execution. If an alternate stage is desired the registration will need to be modified with the Microsoft Plugin Registration tool linked and pictured below. Other custom configuration scenarios may include changing the Execution Mode to 'Asynchronous' or adjusting the Execution Order.

You will also need to use the registration tool if you want to register the plugin to events other than 'Create' or 'Update'. Link to download SDK tools: <https://docs.microsoft.com/en-us/dynamics365/customer-engagement/developer/download-tools-nuget>



The unsecure configuration field contains the GUID identifier for the associated Rule Configuration. If there are additional inrule_RulesEngineAction steps they will all be stored in this configuration field on the InRule Custom Action Step as entity name, ID pairs.



2 Rules Engine Action

The InRule Solution contains a plugin that has a preregistered step called the RulesEngineAction. The InRule Custom Action allows developers to trigger the execution of rules by calling into the Dataverse API. Actions provide an easy way to trigger functionality in Dynamics and can be invoked using multiple different methods. Registering an action creates a corresponding web API endpoint. This endpoint can be used by external code, or in the case of the 'Run Rules' button, called from JavaScript. Actions can also be called from workflows directly with the workflow designer, or through a custom workflow activity.

To call this endpoint, you will need to make a POST request to the following url: https://your_org_url/api/data/v9.0/inrule_RulesEngineAction. The body of the request should contain a JSON object string with the following fields. 'ruleSetName' can be either a rule set or decision name. Label and ruleParameters are optional. If 'label' is not supplied, the default value configured on the execution service will be used. To pass parameters to the rule set or decision, use ruleParameters and provide a JSON string in the following format: {"parameter1Name": "parameter1Value", "parameter2Name", "parameter2Value"}

- entityId
- entityType
- ruleAppName
- ruleSetName
- persistChanges
- label
- ruleParameters

The custom action will return a JSON response with the following schema. This schema can be supplied to JSON parsing steps in Power Automate flows to help with processing rule execution results

```
{ "type": "object", "properties": {
  "NotificationResponse": { "type": "object", "properties": {
    "Notifications": { "type": "array", "items": { "type": "object", "properties": {
      "Type": { "type": "integer" },
```

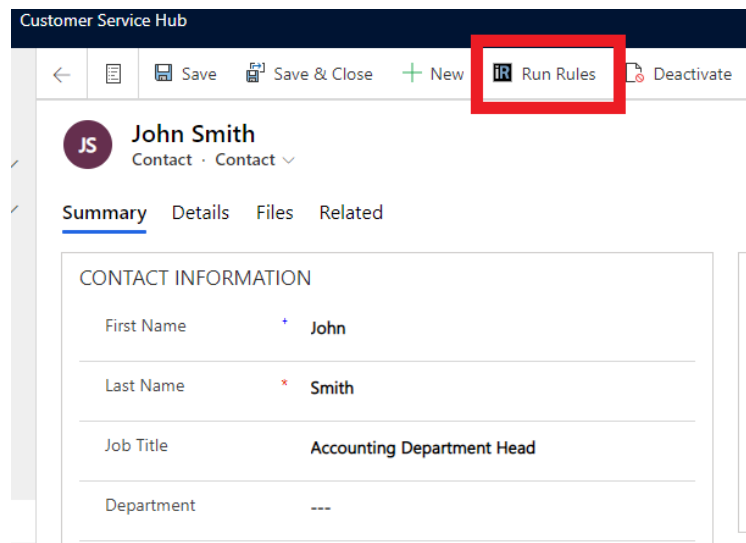
```

    "Message": { "type": "string" } },
    "required": [ "Type", "Message" ] } } } },
    "ValidationResponse": { "type": "object", "properties": {
        "Validations": { "type": "array", "items": { "type": "object", "properties": {
            "FieldName": { "type": "string" },
            "FieldDisplayName": { "type": "string" },
            "Message": { "type": "string" } },
            "required": [ "FieldName", "FieldDisplayName", "Message" ] } } } },
        "ContextResponse": { "type": "object", "properties": {
            "PropertyBag": { "type": "array", "items": { "type": "object", "properties": {
                "key": { "type": "string" },
                "value": { "type": "string" } },
                "required": [ "key", "value" ] } } } },
            "ErrorResponse": { "type": "object", "properties": {
                "Errors": { "type": "array", "items": { "type": "object", "properties": {
                    "Source": { "type": "string" },
                    "Message": { "type": "string" } },
                    "required": [ "Source", "Message" ] } } } } } }
    }
}

```

3 Run Rules Button

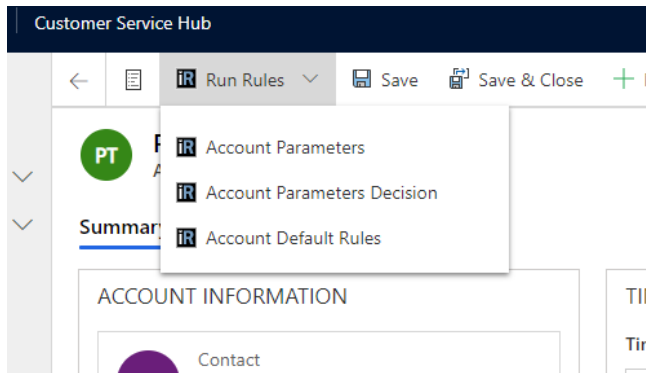
The 'Run Rules' button makes it easy for the end user to choose when to execute rules. Executing rules via the Run Rules button allows for the display of Informational, Warning, and Error notification types, which provide rich feedback to end users. Multiple rule sets can also be configured for interactive selection giving users the option of choosing which rules to run for a given record or dataset. The InRule Solution places this button on the ribbon of each Dynamics 365 or model-driven app Entity Form, as shown here:



When the Run Rules button is pressed, the included JavaScript will execute rules against the current entity based on the settings defined in the InRule Solution – Rules Configuration Form, described in [Appendix F: Rules Configuration and Settings](#). Once you have set up the basic InRule configuration and validated all functionality, you can hide this button from all forms where it is not used to prevent confusion for end users. To do this, follow the steps in [Disabling the InRule Run Rules Button](#) to hide the button in the default configuration, and then create new configurations to show the button only on the entities that have associated rules.

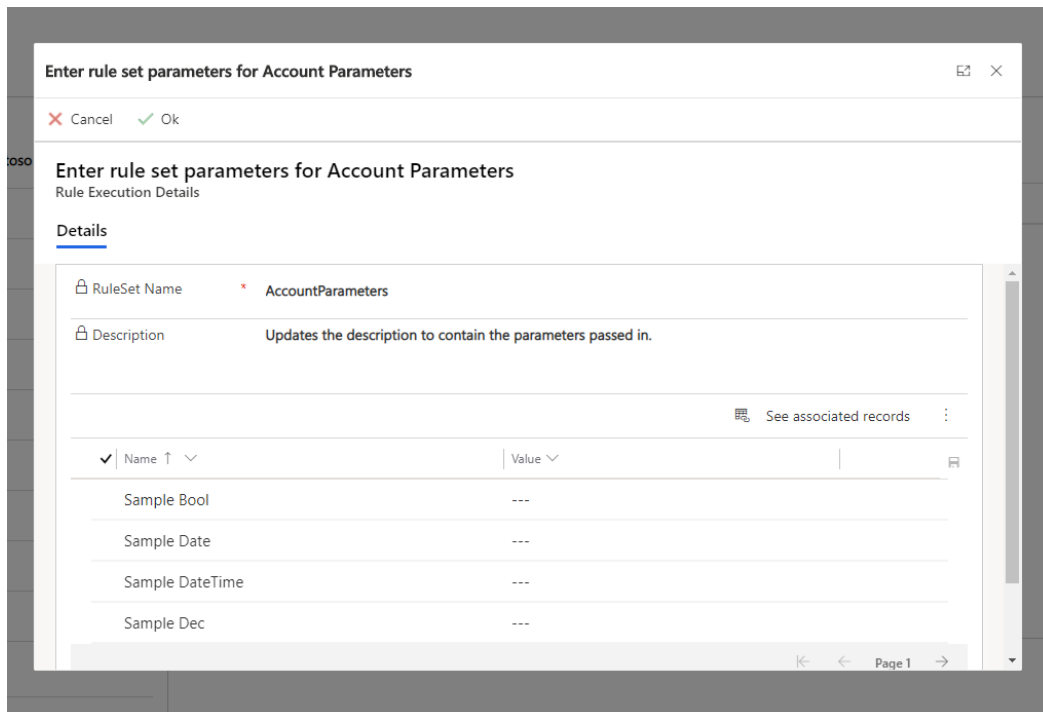
Using the Rule Set Dropdown

Out-of-the-box, the Run Rules button is scoped to only run the default rule set that is specified in the configuration file. However, you also have the option to add additional rule set configurations for selection by the end user. When these are added, the Run Rules Button will display a dropdown menu to allow the end user to select the rule set or decision. To enable this functionality, follow the steps in [Configuring the Run Rules Button for an Entity Form](#)



Running Rules with parameters

If you need to collect information from end users that is not on an entity form, you can configure rule set parameters to get this information at runtime. When parameters are configured for a selected rule set or decision, the end user will see the following dialog to allow them to input the parameters:

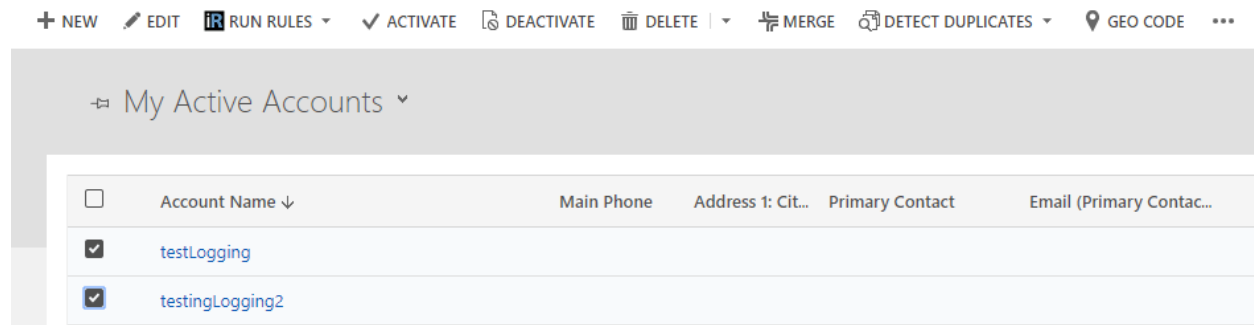


Users can optionally provide values for each parameter by setting the 'Value' field using the editable grid. Once they have filled out the parameters, they can click the 'Ok' command button, and rule execution will continue as usual. If the user clicks the 'Cancel' button or closes out of the dialog for any reason, rule

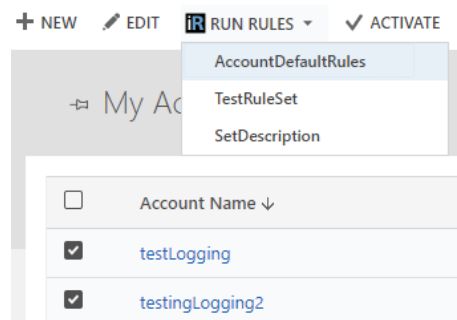
execution will also be canceled. This functionality is primarily intended for use with the 'Run Rules' button, but these parameters can also be passed in directly to the Rules Engine Action, which can be called from flows, JavaScript or other external callers.

Using the Run Rules button from Entity View

The Run Rules button can also be used from an entity view page. This allows for running rules against multiple entities at once. To do so, simply navigate to the entity view for the entity type you wish to run rules against and select the entities against which you want to run rules.



Once at least one entity is selected, a Run Rules button will appear in the Ribbon. This Run Rules button functions identically to the Run Rules button on an entity page and provides a similar dropdown menu for multiple rule sets or decisions associated with an individual rule configuration.



Running rules in this manner will use the rule configuration associated to the entity type that you are running against.

Running rules in this manner will use the rule configuration associated to the entity type that you are running against.

A key limitation to be aware of while running rules via this approach is that entity view pages are unable to display any sort of banner notifications or rule execution status. Should you want to view any notifications associated with a rule execution from a view page, refer to the plugin trace log.

Additionally, the entity view page run rules button will only execute rules against entities loaded on the current page, up to a maximum of 100 entities.

4 Workflow Activity

If you need to run rules from a workflow, the solution provides a custom workflow activity that invokes the included custom action. The custom workflow activity passes in the entity ID and type automatically for


you and provides the rest of the custom action settings as input fields on the activity. The activity also returns the details of the response as individual fields, instead of raw json. These are the fields returned:

- Informational Notifications
- Warning Notifications
- Error Notifications
- Validations
- Errors

When using the custom workflow activity, you can select the option to throw an exception on failure. If you choose to throw an exception on failure, the workflow will end immediately if the custom action fails, and the user will see an error message if they are running the workflow synchronously. If you want to implement conditional logic to handle failures, you can disable this option and check for the existence of data in the Error return variable to determine if the activity failed.

If you don't want to use the default configuration for a particular entity, create a new InRule Configuration record according to [Updating InRule Rules Configuration Records](#) and then associate it with that entity by following [Associating an InRule Configuration record to an Entity](#).


Additionally, if you want to run rules against multiple entities, you can leverage custom workflows from an entity view page and run the workflow against selected entities. This functions similarly to the entity view page Run Rules button, but the notable difference is that custom workflows allow for the execution of custom logic on the selected entities before the execution of rules.

 **Important:** Before you can use the custom workflow activity, you will need to update the max plugin depth setting for the custom action step to be at least 2, instead of the default value of 1. This is because causing a plugin to be run from a workflow adds 1 to the current plugin depth. For more information on plugin depth, please refer to [Changing the Max Plugin Depth](#)

5 Form Events

While the 'Run Rules' button provides an easy way to run rules on demand, you can also run rules automatically on form events, such as 'OnSave'. To help with this, the invokeCustomAction.js web resource provides a helper function called 'executeRulesOnEvent' that you can easily register to an event.

When this function is registered to the 'OnSave' event of a form, it will trigger the rules configured for the entity or event (see specific steps below). Any validation errors returned by the rules will automatically cancel the save operation and display the messages in the notification pane. Alternatively, if you need to use validations to prevent saving for updates made from the Dataverse API, you'll need to configure rules to run on the 'Update' or 'Create' event for an entity as described in [Dataverse Events](#).

 **Important:** If you want to use the dirty field values currently on the form, and not the values already saved to Dataverse when running rules, make sure you set 'Use Dirty Entity Image' to true when setting the InRule Configuration for this entity. You can do this by following the steps in [Updating InRule Rules Configuration Records](#) and [Associating an InRule Configuration record to an Entity](#).

1. Registering a function to a form event requires customizing the entity form. If you want to make this change in a solution navigate to that solution first. Otherwise, navigate to Settings -> Customization -> Customize the System

Dynamics 365

Settings

Customizations

Customization

Which feature would you like to work with?



Customize the System

Create, modify, or delete components in your organization. Components include entities, fields, and relationships.



Solutions

Create, modify, export, or import a managed or unmanaged solution.



Themes

Adjust your organization's colors. Create, change, or delete themes that are used in your organization.

2. Navigate to the entity and form you'd like to customize and select it

Account

Forms

Solution Default Solution

Components

Entities

Account

Forms

Views

Charts

Fields

Keys

1:N Relationships

N:1 Relationships

N:N Relationships

Messages

Business Rules

Hierarchy Settings

Dashboards

Account Project Pricing

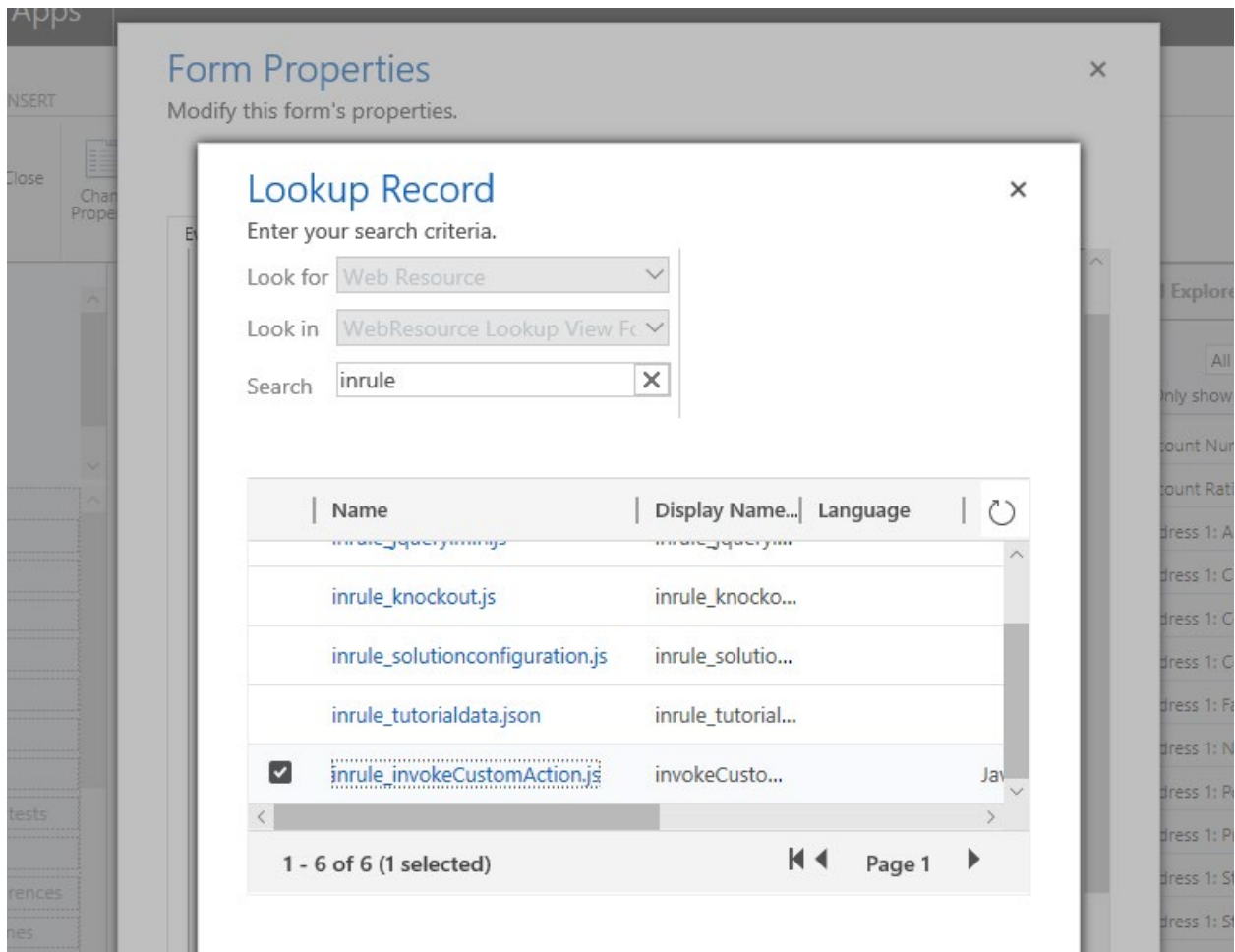
Action Card

System Forms Active Forms

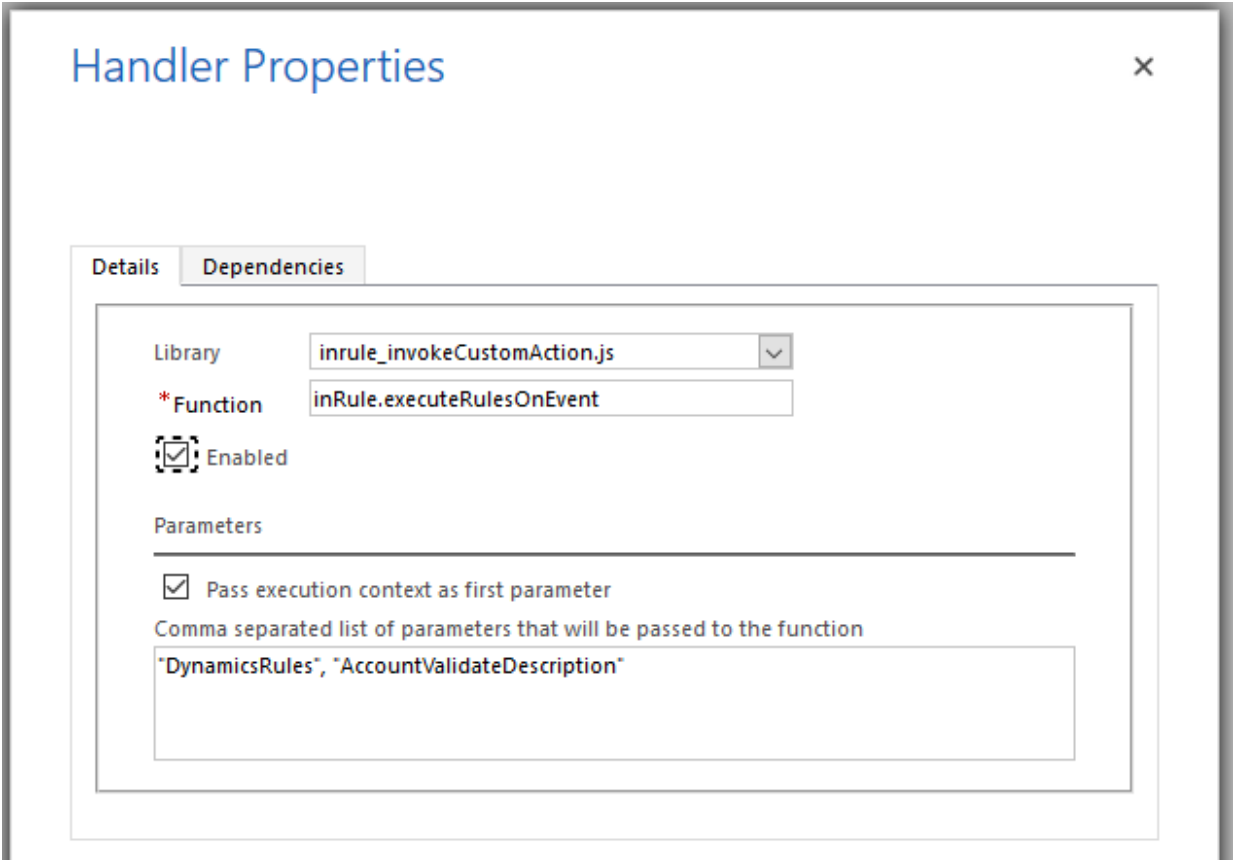
New Delete Enable Security Roles

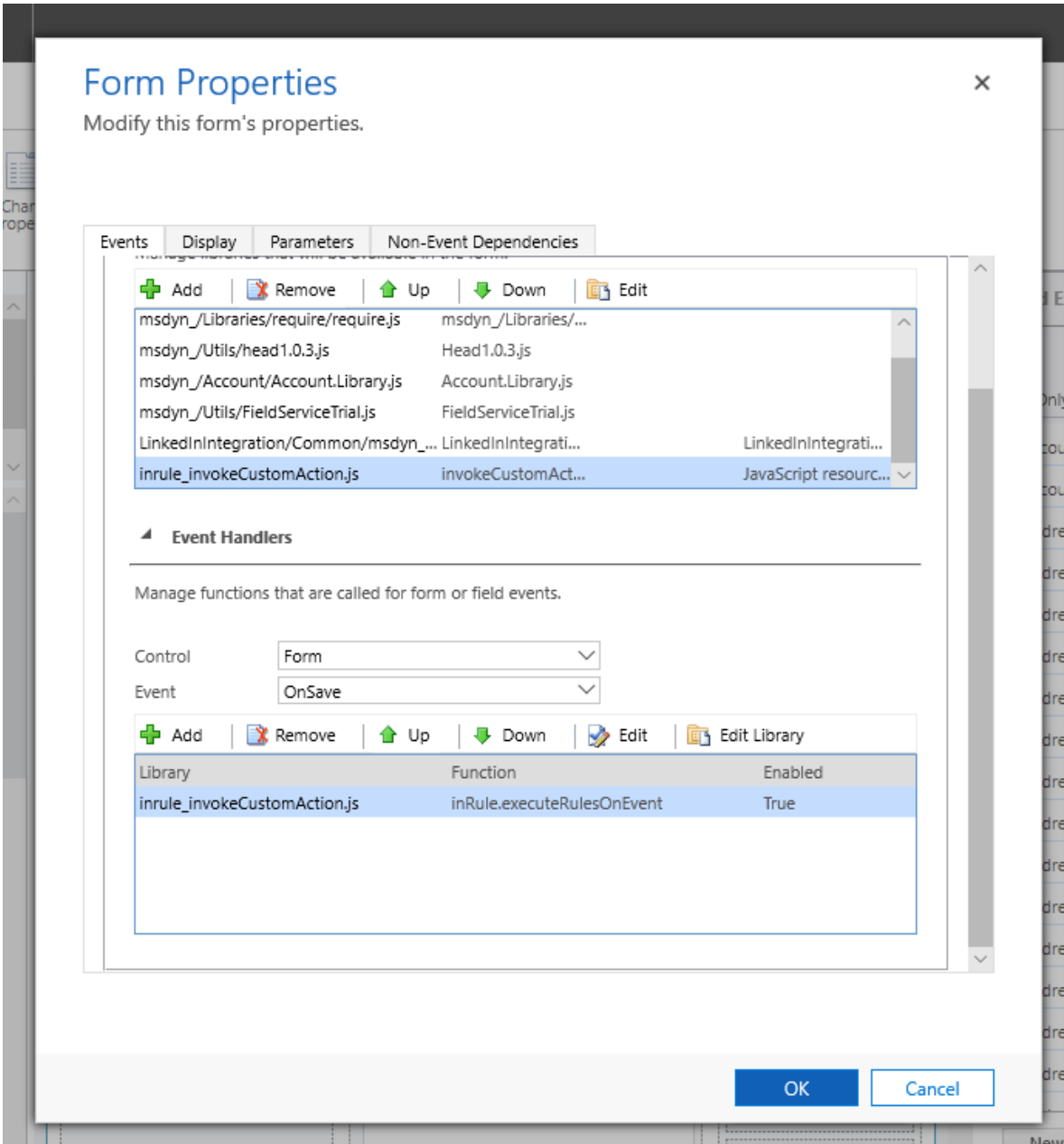
Name	Form Status
Account Card form	Active
Account - MoCa	Active
Sales Insights	Active
Account	Active
Account Quick Create	Active
Account Summary Card	Active

3. Once the form is open, select 'Form Properties' in the ribbon bar, click 'Add' under 'Form Libraries', find `inrule_invokeCustomAction.js` and add it



4. Select the desired Control and Event under 'Form Handlers' and click 'Add'. In the window that pops ups, select 'inrule_invokeCustomAction.js' for the Library, and enter 'inRule.executeRulesOnEvent' for the Function. Under the 'Parameters' section, ensure the 'Pass execution context as first parameter' is checked. If you want to override the default rule set and rule app configured for the entity, the rule app can be passed in as the first parameter, and the rule set or decision as the second. Both must be passed for this to work.





5. Click 'Ok', 'Ok' and then save and publish the form

6 JavaScript

While InRule provides the ability to easily run rules on demand with the 'Run Rules' button and form events, you can also write your own JavaScript for more advanced scenarios and consume the functions provided in the included invokeCustomAction.js resource directly.

Overriding Rule Configuration behavior with user Options and custom JavaScript

The **Run Rules** ribbon button executes the `InRule.executeRules()` method without passing in any parameters. Calling the method in this way will use the configuration defined by the InRule Solution – Rule Configuration Form. However, if an implementer chooses to call the `InRule.executeRules()` method

from within custom JavaScript code, a `userOptions` object can be passed in that will override the Rule Configuration.

The below JavaScript clip demonstrates how to load up a `userOptions` object with the values, and then pass those options on to `inRule.executeRules(userOptions)`. 'ruleSetName' can be either a rule set or decision name. To pass parameters to the rule set or decision, use `ruleParameters` and provide a JSON string using the format below:

```
var formContext = executionContext.getFormContext();
var userOptions = {
    persistChanges: true,
    useDirtyEntity: false,
    entityType: formContext.data.entity.getEntityName(),
    entityId: formContext.data.entity.getId(),
    useEntityPrefix: false,
    showConfirmation: true,
    showStatus: true,
    ruleAppName: "MyRuleApplication",
    ruleSetName: "MyRuleSet",
    ruleSetParameters: '{"parameter1Name":"parameter1Value",
"parameter2Name", "parameter2Value"}'
};
inRule.executeRules(formContext, userOptions);
```

7 Power Platform

Over the years Microsoft has extended the functionality available in Dynamics 365 by providing deep integration into other Microsoft products. Most of these products now fall under the 'Power Platform' umbrella, which includes things like Power Apps and Power Automate. These technologies are underpinned by the Dataverse and the Common Data Model, which is the same technology used to store and manage entities in Dynamics 365. These are some examples of things you can do with InRule and Power Platform:

- **Power Automate** – Use the Dataverse Connector to run rules against an entity as part of a flow, and use the rule output later in the flow
- **Dataverse and Model-driven Apps** – Create custom model-driven apps that execute rules using the Run Rules button and Plugin events, just like in Dynamics 365
- **Canvas Apps** – Using the integration with Power Automate, execute rules from forms with entities to display notifications or take other actions based on rule output
- **Power App portal** – Run rules and display rule output in customer-facing portals (requires custom code for UI and triggering custom action – contact InRule for more information)

In the following sections, we will describe in more detail how to use InRule from both Power Automate flows and Canvas apps.

Power Automate

Power Automate is Microsoft's low-code solution for building workflows and business logic. The primary way Power Automate flows can interact with Dynamics is through the Dataverse Connector. This connector lets you interact with entities, as well as other Dataverse-specific functionality, like invoking a custom action. Using the custom action step, you can call the InRule RulesEngineAction included with the InRule solution, and pass in all the required information to execute rules against an entity. This will kick off the same plugin logic that is used by the Run Rules button and entity events within Dynamics and return the same set of information. If the rules you've written handle making all of the changes you need, you can simply fire the RulesEngineAction and move on, or you can use the rule output later in the flow for downstream processing. You could, for example, write the notifications returned from rule execution to an email and send that to a user. The steps below will show you how to set up a flow to execute rules on an entity.

1. Once you've installed InRule for Dynamics 365, go to <https://flow.microsoft.com>, and make sure you've selected the Dynamics or Dataverse environment containing the InRule solution



Select environment

2. Create a new flow using whichever trigger is appropriate for your flow, and then add a new 'Perform an unbound action in selected environment' action.

Add an action




Runtime

Select a runtime

Action Type

Actions

☒ Group by Connector

 Microsoft Dataverse

[In App](#)

[See more](#)


Perform an unbound action

[In App](#)

Perform an unbound action in selected environment

[In App](#)

3. Select the current environment from the environment dropdown. Next, search for the 'inrule_RulesEngineAction' action in the list of available actions.

 **Perform an unbound action in selected environment** ⋮ <

Parameters

Settings

Code View

Testing

About

Environment *

InRule Technology QA Sandbox

Action Name *

inrule_RulesEngineAction

4. Once you select the action, the step should automatically create the required fields for the RulesEngineAction. To view the fields, hit the 'Show all' button next to the Advanced Parameters drop down. Most of these values will typically be static, but the entity ID will need to be provided as an input to the flow, or read from an entity object used in the flow

Advanced parameters

Showing 8 of 8

Show all

Clear all

Item/EntityId

9ebbb704-345c-eb11-bb23-000d3a55da4f

×

Item/EntityType

account

×

Item/RuleAppName

DynamicsRules

×

Item/RuleSetName

AccountDefaultRules

×

Item/PersistChanges

Yes

×

Item/DirtyEntityImage

×

Item/RuleParameters

×

Item/RuleAppLabel


×

- If you want to use the output of the rule execution in your flow, you will likely need to parse the JSON result to access the individual components of the response. In most cases, this should not be necessary because changes are automatically saved back to Dynamics. To do this, add the 'Parse JSON' step, and select the 'executionResult' content from the unbound action step. In the 'schema' field, copy and paste the JSON schema from the [Custom Action](#) section.

The screenshot displays the 'Parse JSON' application interface. At the top, there is a header bar with a logo on the left and a menu icon on the right. Below the header, there are four tabs: 'Parameters', 'Settings', 'Code View', and 'About'. The 'Parameters' tab is currently selected and highlighted with a blue underline. Below the tabs, there is a section labeled 'Content *' which contains a single entry with a green icon and the text 'executio...'. Below this, there is a section labeled 'Schema *' which displays a JSON schema. The schema is a large object with two main properties: 'NotificationResponse' and 'ValidationResponse'. Each of these properties is an object with its own set of properties, including 'type', 'properties', 'required', and 'items'. The 'NotificationResponse' object has properties for 'Notifications', 'Type', 'Message', and 'required'. The 'ValidationResponse' object has properties for 'Validations', 'FieldName', 'FieldDisplayName', 'Message', and 'required'. The 'ContextResponse' object has properties for 'PropertyBag', 'key', 'value', and 'required'. The JSON is color-coded, with strings in blue, integers in orange, and arrays in green.

Use sample payload to generate schema

6. Working with nested objects can be difficult in Power Automate, so if desired you can add an additional step to help break out the values you want. If, for example, you want to use notifications and errors, you can parse the initial response as usual in the first step, and then have additional parse JSON steps to parse out notifications and errors.

 **Parse Errors**


Parameters

Settings

Code View

About

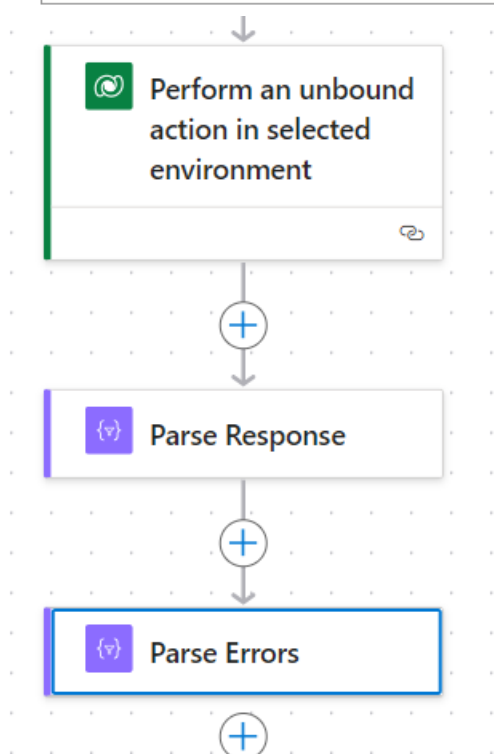
Content *

 Body Err... x

Schema *

```
{ "type": "array", "items": { "type": "object", "properties": {  
  "Source": { "type": "string" },  
  "Message": { "type": "string" } },  
  "required": [ "Source", "Message" ] } }
```

[Use sample payload to generate schema](#)



```
graph TD; A[Perform an unbound action in selected environment] --> B[Parse Response]; B --> C[Parse Errors];
```

If you follow this method, you can select the following properties from the output of the initial step, then use the provided JSON schemas to parse just that property:

a. Notifications

```
{ "type": "array", "items": { "type": "object", "properties": {
```



```
"Type": { "type": "integer" },
"Message": { "type": "string" } },
"required": [ "Type", "Message" ] } }
```

b. Errors

```
{ "type": "array", "items": { "type": "object", "properties": {
  "Source": { "type": "string" },
  "Message": { "type": "string" } },
"required": [ "Source", "Message" ] } }
```


c. Validations

```
{ "type": "array", "items": { "type": "object", "properties": {
  "FieldName": { "type": "string" },
  "FieldDisplayName": { "type": "string" },
  "Message": { "type": "string" } },
"required": [ "FieldName", "FieldDisplayName", "Message" ] } }
```

d. PropertyBag

```
{ "type": "array", "items": { "type": "object", "properties": {
  "key": { "type": "string" },
  "value": { "type": "string" } },
"required": [ "key", "value" ] } }
```


- If you need to return results from this flow to, for example, a canvas app, you can either build your own response object, or you can feed the execution result directly into the 'Response' step, using the same content and schema as above


Response
⋮
<

Parameters
Settings
Code View
About

Status Code *

Headers

Enter key	Enter value	
-----------	-------------	---

Body

Advanced parameters

Showing 1 of 1

Show all

Clear all

Response Body JSON Schema

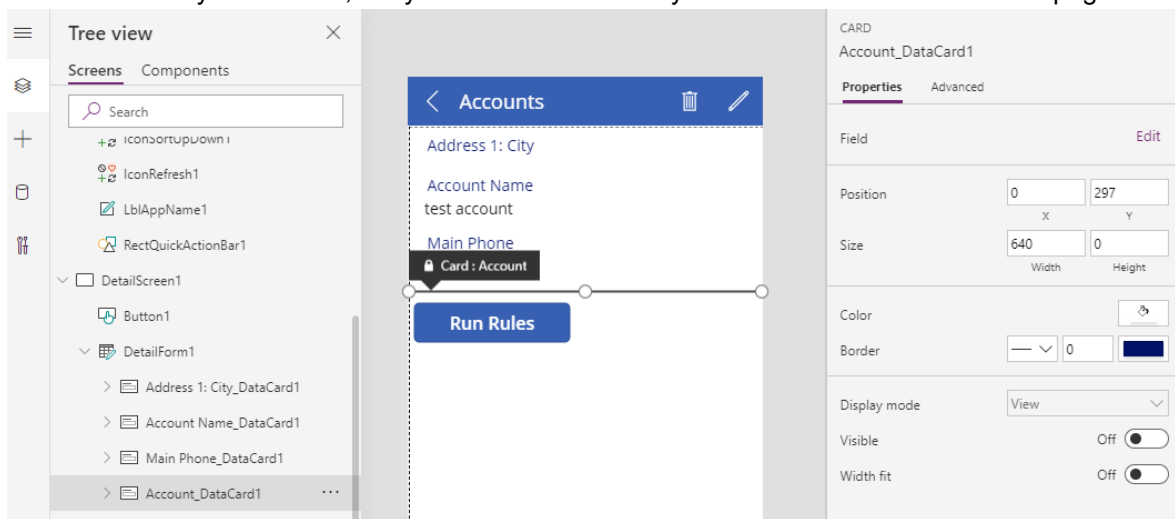
```
{
  "type": "object",
  "properties": {
    "NotificationResponse": {
      "type": "object",
```

✕

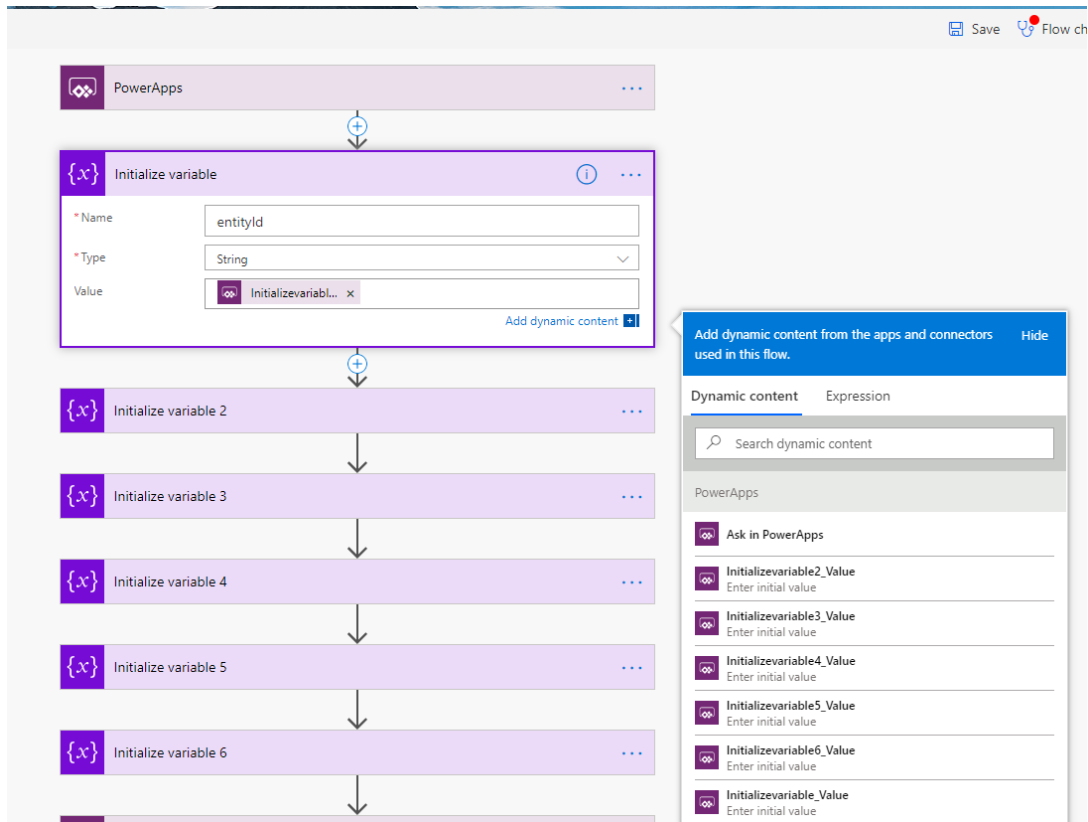
Canvas Apps

While model-driven apps like Dynamics 365 are useful in many scenarios, canvas apps provide more flexibility in design. While canvas apps do not necessarily have to use Dataverse entities, if your app uses entity forms or integrates with entities in some other kind of way, you can utilize the integration with Power Automate to run rules against these entities and display information from rule output directly in your app. We will provide some boilerplate code here for potential use in a canvas app, but given the flexibility of canvas apps, your code may look very different. The following directions will show you how to add a 'Run Rules' button to a page with an entity form and use the output from rule execution to display notifications to the user.

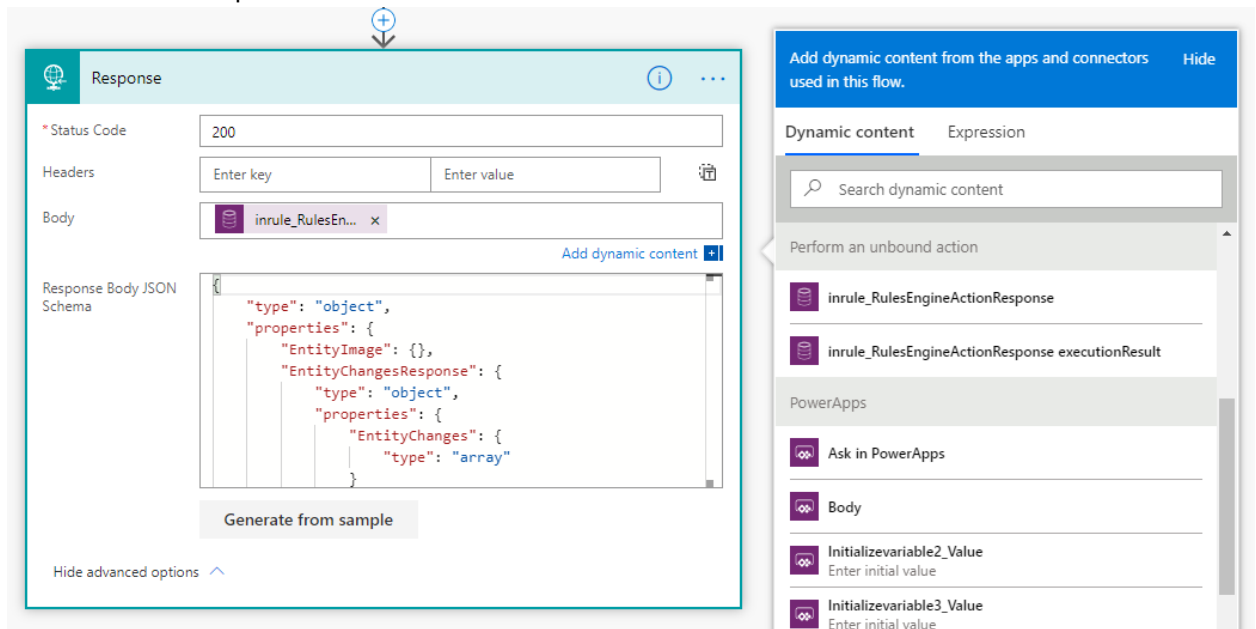
1. The starting point for these steps is a canvas app that already has a screen with an entity form on it. In order to run rules against the particular entity the user is viewing, you will need to add the ID field for the entity to the form, but you can set the visibility to 'Off' so it won't show on the page



2. Next, we'll need to create a flow that will invoke the included RulesEngineAction and run the rules. For more information on how to set up a basic flow to run rules, refer to the previous section on Power Automate. When creating this flow, you will need to use the 'Power Apps' trigger. This will make the flow available for selection and allow for input from the canvas app. To get input from the canvas app, click in a content field and choose 'Ask in Power Apps' from the 'Power Apps' trigger. This will automatically create a new content variable that you can reference in the flow and make that variable available as input to the flow. You can add as many of these variables as you want, but you will need to add at least one variable for the entity ID. Other values required by the RulesEngineAction, such as rule app or rule set name, can either be hard coded into the flow, or passed in via variables

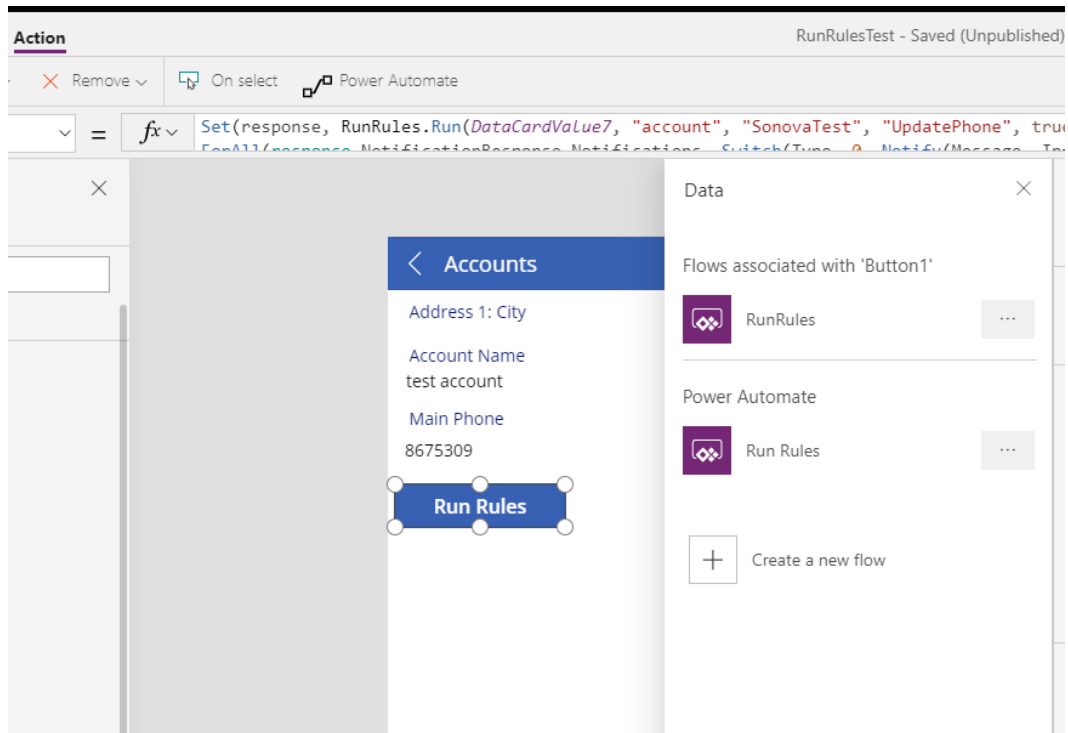


3. If you want to use the execution result in your canvas app, you'll need to add a 'Response' step to the end of the flow. The 'executionResult' will need to be returned in the body, and the JSON schema from the [Custom Action](#) section will need to be provided in the 'JSON Schema' section under 'Advanced Options'



4. Next, you'll need to add a button and link it to Power Automate flow we just created. To do this, add a new button to the screen and go to the 'Action' tab for the button. Choose 'Power Automate' and you should see the flow you created in the previous step listed here. If you do not

see the flow, make sure you canvas app and flow are in the same environment. Select your flow to associate it to the button



5. You can use the following function snippet as a starting point for the function for your button. This snippet calls the associated flow (in this case called 'RunRules') and sets the output to a variable. It then loops over all of the notifications in the response, displays them to the user, and refreshes the form data. Due to limitations in the way notifications work in canvas apps, only the most recent notification is shown, but you can expand on the sample with your own custom code. In this example, all of the properties required for the RulesEngineAction are passed in to the flow, but you may choose to set these in the flow itself. The only value that must be passed in is the entity ID, which is the first parameter here. When editing this code in the function editor, the intellisense will display which parameter maps to which variable in the flow.

```
Set(response, RunRules.Run(AccountIdValue, "account", "DynamicsRules", "UpdatePhone", true));

ForAll(response.NotificationResponse.Notifications,
Switch(Type,
0, Notify(Message, NotificationType.Information),
1, Notify(Message, NotificationType.Warning),
2, Notify(Message, NotificationType.Error)));

Refresh(Accounts);
```

```
Run(Initializevariable_Value, Initializevariable2_Value, Initializevariable3_Value, Initializevariable4_Value, Initializevariable5_Value)

Set(response, RunRules.Run(AccountIdValue, "account", "SonovaTest", "UpdatePhone", true));
ForAll(response.NotificationResponse.Notifications, Switch(Type, 0, Notify(Message, Notifi
```

6. Once the setup is complete, you can test the rule execution by previewing the app and hitting the 'Run Rules' button. If execution is successful, you should see any notifications from rule execution displayed in the app

Appendix F: Rules Configuration and Settings

The *InRule Solution – Rules Configuration Form* provides a central location where you can configure rule execution options across the various methods of executing rules. By leveraging these Rule Configurations users can configure the [Run Rules Button](#) and events to execute Rule Sets or Decisions for a given entity, which provides the ability to run complex logic against that entity and its related entities.

At a summary level, the steps for setting up the Run Rules Button are as follows:

- [Create a Rule Configuration Record](#)
- [Configure Rules Button Settings](#)
- [Create a Rule Set Configuration for each Rule Set or Decision](#)
- [Associate a Rule Configuration Record to an Entity](#)

The Run Rules Button is also able to take advantage of Rule Sets or Decisions that utilize parameters, more information about which can be found in:

- [Using Input Parameters for Rule Sets or Decisions](#)

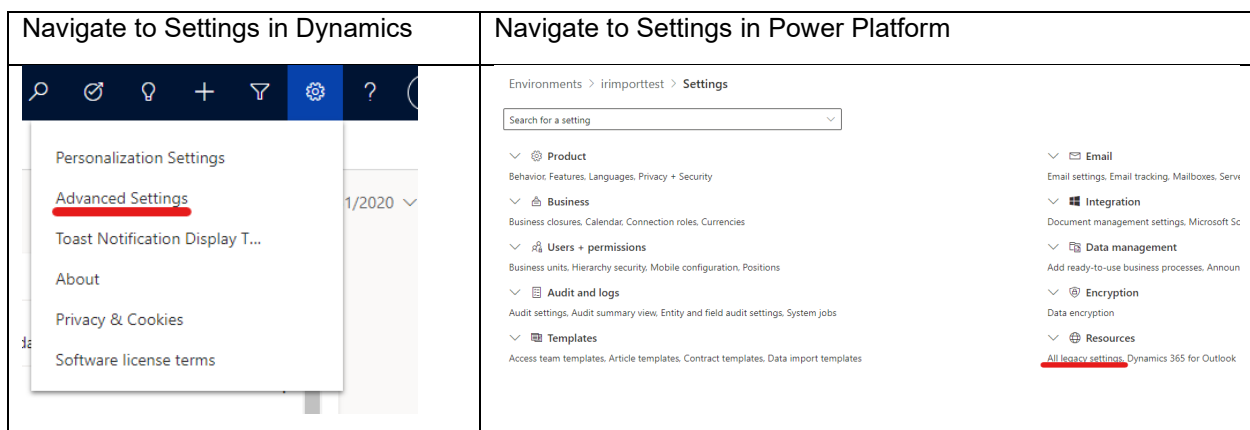
Rule Configurations can also be configured to run a Rule Set or Decision against an entity on certain [events](#) (entity create or update). The primary areas of focus for associating a Rule Set or Decisions with an event are:

- [Update a Rule Configuration Record](#)
- [Associate a Rule Configuration Record to An Entity Event](#)

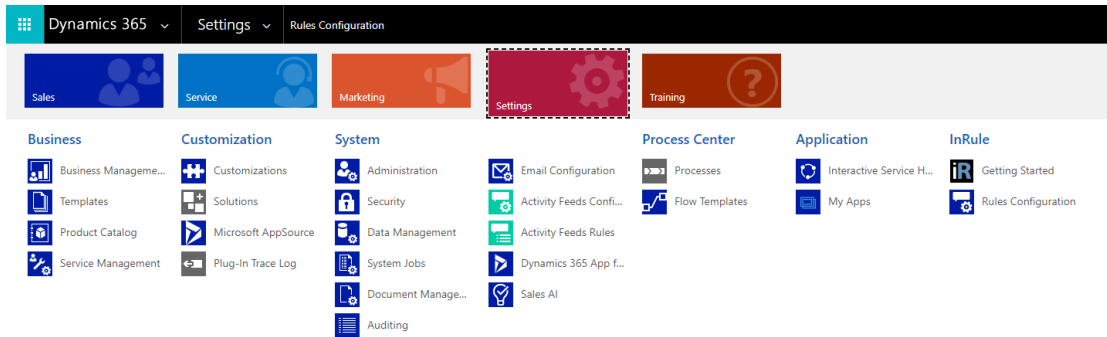
These events only use the primary Rule App and the primary Rule Set or Decision that are configured on the Rule Configuration, so the Rule Set Configurations and Parameter Configurations that can be used with the Run Rules Button are not necessary.

Create or Update Rule Configuration Records

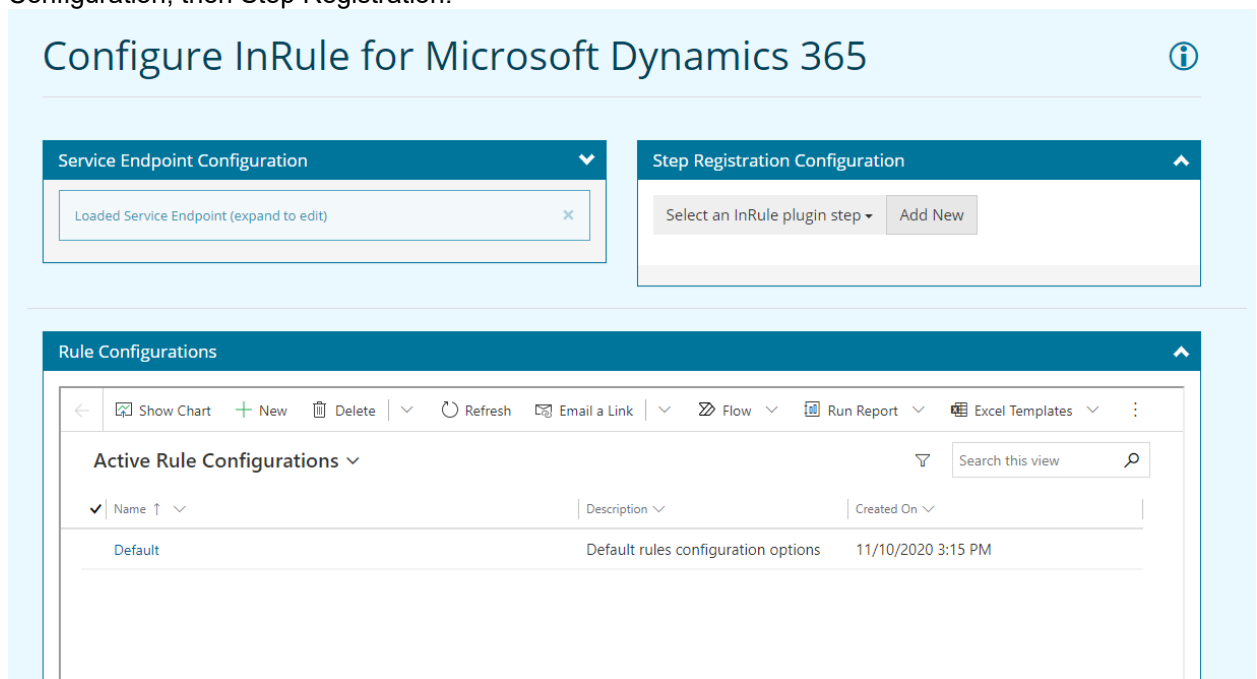
1. If you are already logged in to Dynamics 365 or a Power App, click on the settings cog in the top right and go to 'Advanced Settings'. Alternatively, if you are in Power Platform Admin center, go to the settings for your environment and then click on 'All legacy settings'. From the settings menu, select Rules Configuration under the InRule section.



- From the settings menu, click on 'Rules Configuration' under the 'InRule' section



- The Rules Configuration page contains 3 panes – Service Endpoint Configuration, Step Registration Configuration, and Rules Configuration. The following steps will focus on Rules Configuration, then Step Registration.



- Either add New or select an existing Rule Configuration record (eg Default). The following steps use the Default record, but since Default is more of a global Rule Configuration, you should consider creating a new Rules Configuration specific to the entity that you want to associate to the Run Rules button. Select either a new or existing Rules Configuration record to edit:

← Save Save & Close + New Deactivate Delete Refresh Check Access Flow ⌵ ⋮ Share ⌵

Default - Saved 4/24/2024 6:08 PM Active
Rule Configuration Modified On Modified By Status ⌵

General Rules Button Settings Plugin Settings Notes Related ⌵

General

Name *	Default
Description	Default rules configuration options
Rule App Name *	DynamicsRules
Rule Set Name	DefaultRules

Rule Execution Service


Service Endpoint Id (or Uri) *	d2e50ca1-ef4c-e611-80e9-6c3be5a82b30
Rule Execution Log Enabled *	No
Entity Loading Log Enabled *	No
Return Full Entity Image *	No
AppDomain Cache *	0

Clear Cache

5. Configure the **General Settings** for the Rule Configuration as follows:
 - a. Choose a Name
 - for a Rule Configuration that will be used for a specific entity (which is typically the case), it's best to create a name that represents the entity along with the purpose of the rules (e.g. LoanEligibilityConfg)
 - b. Enter a Description
 - c. Enter the Rule App containing the Rule Set or Decision
 - ! This setting is used as a global/default setting when individual Rule Set Configurations are not provided in the Rules Button Settings section. If the Rule Configuration is being associated to an entity event, then use this setting. If the Rule Configuration is being used for the Run Rules Button, then use the Rule Set Configurations in the Rules Button Settings section and this setting can be left empty.
 - d. Enter the Rule Set or Decision to execute
 - ! Similar to the Rule App setting, this setting is used as a global/default setting when individual Rule Set Configurations are not provided in the Rules Button Settings section. The same logic applies for when to set this field as compared to when to use the Rule Set Configurations ([described here](#)).
 - e. Press "Save" in the top-left corner

Configure Rules Button Settings

Out-of-the-box, the Run Rules button is scoped to only run the default rule set that is specified by the Rule Set Name field on the 'Default' Rule Configuration. However, you also have the option to add additional Rule Set Configurations to allow one or more rule sets/decisions to be selected from the Run Rules Button. When these are added, the Run Rules Button will display a dropdown menu to allow the end user to select from a list of rule sets and decisions. To enable this functionality, follow the steps below:

1. Create or update a Rule Configuration according to the [previous step](#).
 Unless you want this list to appear on all entities, do not modify the 'Default' Rule Configuration record.
2. The **Rules Button Options** can typically be left as the default values, for a complete overview of these settings go to [An Explanation of the Rules Button Options](#).
3. Under **Rules Button Settings**, click the 'Add Existing Rule Set Configuration' button on the 'Rule Set Configurations' sub-grid. From here, you can search for an existing Rule Set if you have already created one as part of another configuration, or you click 'New Record' to create a new record

Default
Rule Configuration


1/25/2021 11:09 AM Modified On | Luke Colburn Modified By | Active Status

General **Rules Button Settings** Plugin Settings Notes Related

Run Rules Button Options

Show Run Rules Button	+	Yes
Use Dirty Entity	+	No
Use Entity Prefix	+	Yes
Show Confirmation	+	Yes
Show Status	+	Yes

Rule Set Configurations

 Add Existing Rule Set ...

✓ Rule Set Name	Display Name	Rule App Name	Sort Order
No data available.			

Page 1

4. If you choose to create a new record, fill out the fields on the form that pops up. Rule Set Name, Type and Rule App are required. Optionally, you can provide a Display Name, which will be displayed in the dropdown instead of the Rule Set Name. Sort Order is also optional and takes an integer which will be used to sort the Rule Set list in ascending order.

New Rule Set Configuration

General

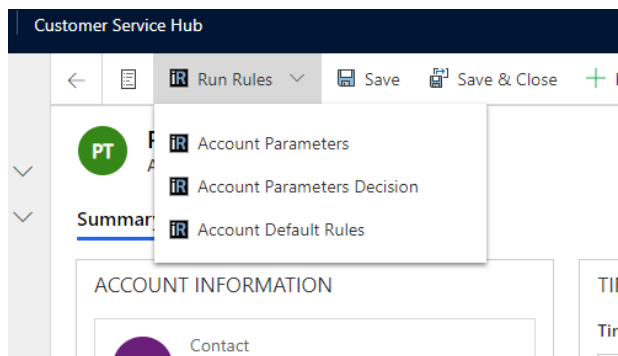
RuleApp Name	*	---
RuleSet Name	*	---
Display Name		---
Type	*	---
Description		---
Sort Order		---

Parameters

Important: When running rules using a Decision, the decision authored in irAuthor must have an input parameter with the entity type you are running rules against. Additionally, any output parameters are not used by the framework, as the same entity is used as output.

This root entity parameter is provided by the framework automatically and does not need to be defined in the Rule Set Configuration page (only in the Rule Application Decision definition). For more information on how to configure other input parameters for your rule set or decision, refer to [Using Input Parameters for Rules](#).

- Associate the configuration record above with the entity you'd like the list to appear on by following the steps in [Associating an InRule Configuration record to an Entity](#)
- Go to the entity page for the configuration that you have just edited and ensure that the Run Rules button now has the desired rule set or decision options listed.

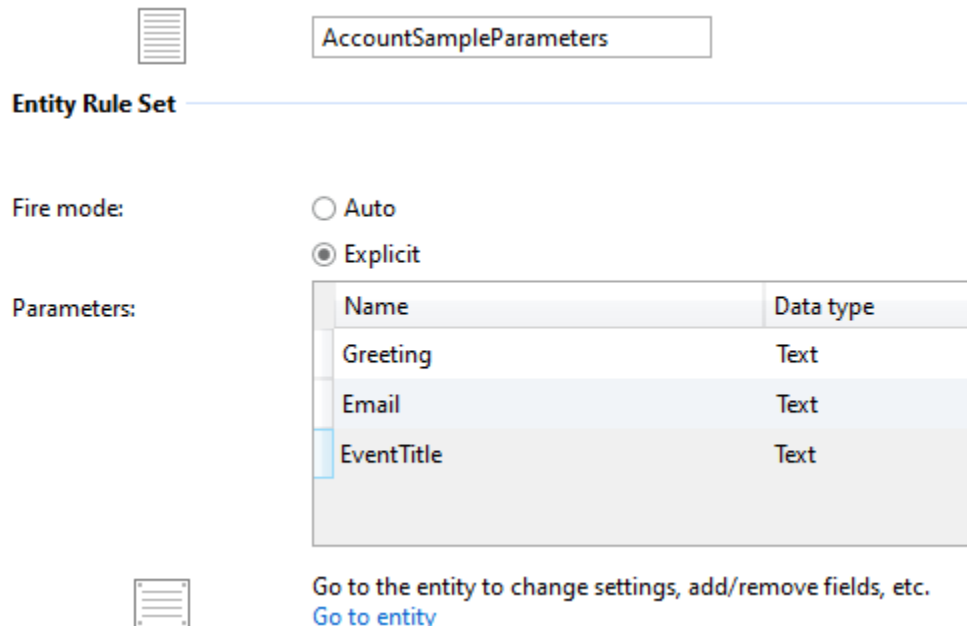


Using Input Parameters for Rule Sets or Decisions

If you need to collect information from end users that is not on an entity form, you can use rule parameters for the user to enter at runtime. If you configure a rule set or decision to use parameters, when a Dynamics user clicks on this rule for execution, they will get a dialog to allow them to input the parameters.

Important: This functionality is not supported in on-prem environments, and is not intended for use with Dynamics events.

To enable this behavior, you will first need to add the parameters to the rule set or decision in irAuthor. InRule for Dynamics supports all the primitive parameter types, but not entity parameters. However, these are all collected as string values and converted in the execution service, so the end user must ensure the values are formatted properly when providing them:



Name	Data type
Greeting	Text
Email	Text
EventTitle	Text

Go to the entity to change settings, add/remove fields, etc.
[Go to entity](#)



Important: When defining a Decision, the decision authored in irAuthor must have an input parameter with the entity type you are running rules against. Additionally, any output parameters are not used by the framework, as the same entity is used as output.

Next you will need to edit the Rule Set Configuration record to use this rule set or decision, along with its associated parameters. In order to get to this point, you should have already gone through these steps:

1. First set up a config record if you haven't already done so according to [Updating InRule Rules Configuration Records](#).
2. Next add an associated rule set config record following the steps in [Configuring the Run Rules Button for an Entity Form](#).

After filling out the form fields and saving the record, you'll now be able to add parameter config entries in the **Parameters** sub-grid.

Example

Rule Set Configuration

General Related

Rule App Name	*	Example
Rule Set Name	*	Example
Display Name		This Is An Example
Type	*	Rule Set
Description		Example
Sort Order		1

Parameters

+

New Parameter Config...

:

✓

Name ↑ ↓

RuleSet Parameter

When adding a parameter, you only need to provide the parameter name, and optionally a description. Repeat this process for each parameter you want to add. Once parameters are added to a rule set configuration, the 'Run Rules' button will automatically display the parameter input dialog to users.

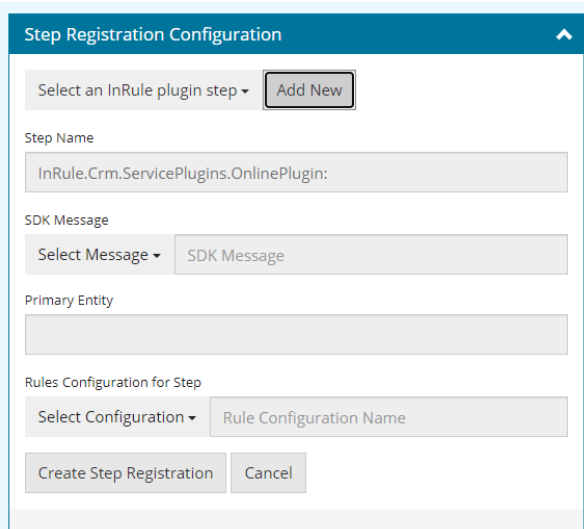
Associating a Rule Configuration record to an Entity

If you want to use different Rule Configurations for different Dynamics entities, you can create multiple configuration records and then associate them to a particular entity. The same configuration record can be associated to multiple different entities if desired. For Rule Configurations you wish you use with the Run Rules Button, `inrule_RulesEngineAction` will be the SDK message you select.

Please note that if you wish to associate an `inrule_RulesEngineAction` to a specific entity type, this will override the "default" custom action step. The "default" `inrule_RulesEngineAction` is a "global" custom action step registration which all entities will default to using until you register a custom action step for a specific entity type.

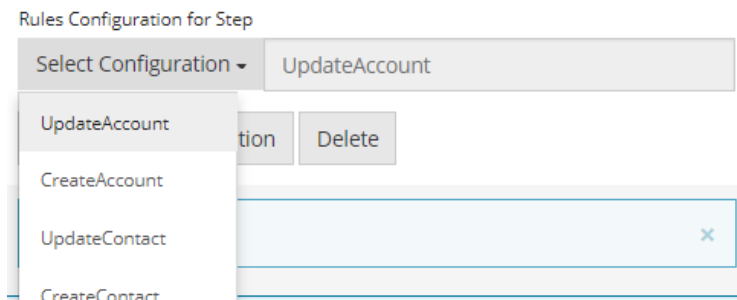
1. Create or update the configuration record according to the [previous steps](#)

2. Return to the Rules Configuration page and navigate to the Step Registration Configuration section and click **Add New**



The screenshot shows the 'Step Registration Configuration' form. At the top, there is a dropdown menu labeled 'Select an InRule plugin step' and an 'Add New' button. Below this, the 'Step Name' field is populated with 'InRule.Crm.ServicePlugins.OnlinePlugin:'. The 'SDK Message' section has a dropdown labeled 'Select Message' and a text field containing 'SDK Message'. The 'Primary Entity' field is empty. The 'Rules Configuration for Step' section has a dropdown labeled 'Select Configuration' and a text field containing 'Rule Configuration Name'. At the bottom, there are two buttons: 'Create Step Registration' and 'Cancel'.

3. Choose the SDK Message (Update, Create, inrule_RulesEngineAction)
 - a. Update – Update of the primary entity
 - b. Create – Create of the primary entity
 - c. inrule_RulesEngineAction – The custom action message invoked when the 'Run Rules' button is clicked. Choose this to override the default behavior for a particular entity whenever the 'Run Rules' button is clicked, custom JavaScript is executed using the included invokeCustomAction.js resource, the workflow activity is used, a form event is used, or the custom action is invoked through some other means.
4. Select a Primary Entity for your rule registration to use
5. Open the Rule Configurations for Step dropdown menu that appears and select the Rule Configuration that you made above.



The screenshot shows the 'Rules Configuration for Step' dropdown menu. The dropdown is open, showing a list of configurations: 'UpdateAccount', 'CreateAccount', 'UpdateContact', and 'CreateContact'. The 'UpdateAccount' configuration is selected, and its details are visible in the background, including a 'Select Configuration' dropdown, a text field with 'UpdateAccount', and buttons for 'Update', 'Delete', and 'Cancel'.

6. Click "Update Step Registration" and verify that the registration successfully saves

Rules Configuration for Step

Select Configuration ▾

UpdateAccount

Update Step Registration

Delete

Saved changes

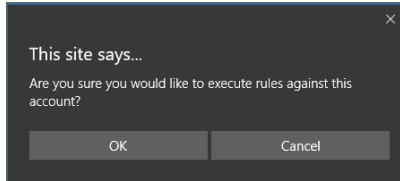
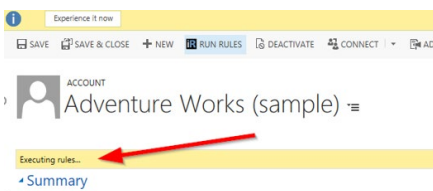
✕

An Explanation of the Rules Button Options

The below options must be passed into the InRule Custom Action. InvokeCustomAction.js exposes InRule.executeRules() which will use values defined in the InRule Solution – Configuration Form. Alternatively, a consumer of InRule.executeRules() can pass in an object that provides specific values that should be used on a call by call basis

General Rules Button Settings Plugin Settings Notes Related

Run Rules Button Options		
Show Run Rules Button	+	Yes
Use Dirty Entity	+	No
Use Entity Prefix	+	Yes
Show Confirmation	+	No
Show Status	+	Yes

Show Run Rules Button	<p>If true, then the <i>Inrule Run Rules Button</i> will be visible on all pages.</p> <p>If false, then the <i>InRule Run Rules Button</i> will be globally removed. This will only effect the the use of the button, but not other registered plugin steps.</p>
UseDirtyEntity	<p>If true, the InRule Custom Action will expect that form data will be included in the API call. The form data is used to populate the root entity in the Rule Engine. This is useful in scenarios when data has changed on the Dynamics form but the save button has not yet been pressed. This allows a user to run rules against data before a save takes place.</p> <p>If false, the only data passed to the Custom Action is the Entity ID (guid) of the entity, and the Custom Action will query Dynamics directly for that entities data. This limits the InRule Custom Action to being aware only of data that has been fully saved to Dynamics.</p>
UseEntityPrefix	<p>If true, then the supplied <i>RuleSetName</i> is interpreted as a suffix to the <i>EntityType</i> Name. For example: if the supplied <i>RuleSetName</i> is "DefaultRules" and the <i>EntityType</i> Name you are dealing with is "Account", then the <i>RuleSetName</i> actually used will be "AccountDefaultRules".</p> <p>If false, then the supplied <i>RuleSetName</i> is interpreted literally.</p>
ShowConfirmation	<p>If true, this will ask the user in the User Interface with the following visual prompt before actually executing rules:</p> 
ShowStatus	<p>If true, a status messages will be displayed within the Dynamics interface when rules are executing, when rules are finished, and if rules are cancelled by the confirmation dialog.</p> 

Displaying Additional Logs and Response Data in Trace Logs

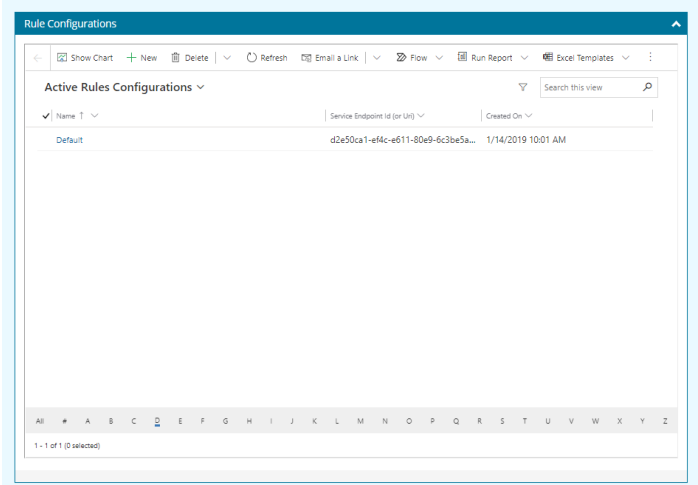
The rule execution service generates logs that are helpful for troubleshooting rule execution, but by default these are only logged on the execution service itself and not returned in the response to reduce response size and avoid trace log truncation. However, for convenience different log sections can be configured to be returned in the response and logged to the plugin trace log. These sections are:

1. Rule Execution Log: The flow of rules and changes in state during rule execution
2. Entity Loading Log: What and how many entities are loaded on the rule execution service before rule execution, and how long it takes to retrieve these entities from Dynamics. More details can be found in [Execution Service Performance](#)
3. Entity Image: The complete root entity in JSON format including changes from rule execution

When enabling these options as much as possible will be logged, but due to character limits in the plugin trace log some data may be truncated. By default all of these options are all off, but they can be enabled with the following steps:

1. Navigate to the InRule Solution Rule Configuration Form

- 2. Scroll down and select the 'Default' record under the 'Rule Configurations' section. Please note that the change made in this specific record will take effect across the whole system.



- 7. Switch to 'Yes' any of the logs you'd like to be returned under the **Rule Execution Service** section and save the config record:

← Save Save & Close + New Deactivate Delete Refresh Check Access Flow Share

Default - Saved 4/24/2024 6:08 PM Modified On Modified By Active Status

General Rules Button Settings Plugin Settings Notes Related

General

Name	* Default
Description	Default rules configuration options
Rule App Name	+ DynamicsRules
Rule Set Name	DefaultRules

Rule Execution Service

Service Endpoint Id (or Uri)	* d2e50ca1-ef4c-e611-80e9-6c3be5a82b30
Rule Execution Log Enabled	+ No
Entity Loading Log Enabled	+ No
Return Full Entity Image	+ No
AppDomain Cache	+ 0

Clear Cache

Disabling the Run Rules Button

Out-of-the-box, the Run Rules button is used to manually initiate the execution of rules. In some instances, the Run Rules button may not be needed, and the button can be disabled as described below.

- 1. Create or update a Rule Configuration according to [Create or Update Rules Configuration Records](#). If you want to hide the button for all forms on all entities, edit the default record. Otherwise, create a new record for the specific entity you'd like to show or hide the button on.
- 2. Set the **Show Run Rules** Button under the **Rules Button Settings** section

General **Rules Button Settings** Plugin Settings ...

Run Rules Button Options

Show Run Rules Button	+	No
Use Dirty Entity	+	No
Use Entity Prefix	+	Yes
Show Confirmation	+	Yes
Show Status	+	Yes

3. If you edited the default record, you can skip this step. Otherwise, you'll need to associate the Configuration record you just modified with a particular entity by following the steps in [Associating an InRule Configuration record to an Entity](#)

Changing the Max Plugin Depth

Dynamics provides the plugin depth property to the plugin context to track the current depth of the call stack. Whenever another plugin or process is executed within the scope of the current transaction, this value is incremented by 1 in the invoked plugin. This value is commonly used to prevent infinite loops, where multiple plugins keep firing each other. The Max Plugin Depth setting in the InRule Configuration will stop execution of the plugin whenever the depth is greater than the specified value. By default, this value is set to 1, but you may need to change it in certain cases, such as using the included custom workflow activity, or running rules from within your own custom plugin. To update this value, change the 'Max Plugin Depth' setting using the steps in [Updating InRule Rules Configuration Records](#), and optionally associate the record with a specific event and entity by following the steps in [Associating an InRule Configuration record to an Entity](#)

Disabling Persist Changes

When Persist Changes is set to enabled all changes made to all entities by InRule are persisted back to Dynamics. When disabled any changes made by InRule are sent back in the response and can be consumed by the caller, but they are not persisted back to Dynamics.

The Persist Changes setting can be found in the Plugin section of the Rule Configuration.

Configuring the AppDomain Cache

When using the Rule Helper, the integration framework provides the ability to save query results in a persistent cache in the execution service AppDomain. This is useful if you have relatively static data that is used consistently in one or more rule apps, since this cache persists across rule executions. By default, this cache is disabled, but you can enable it by setting a value for the cache timeout in the Rule Configuration. To update this value, change the 'AppDomainCache' setting to the desired retention time in seconds using the steps in [Create or Update a Rule Configuration Record](#), and optionally associate the record with a specific event and entity by following the steps in [Associating a Rule Configuration record to an Entity](#).

The AppDomain Cache can be cleared prior to its configured duration in three ways:

- The Clear Cache button, available right underneath the configuration field
- Resetting the cache value to zero. This causes the cache to be cleared the next time the rule helper is used from the execution service
- Restarting the Rule Execution Service

For more information on how to use the rule helper, refer to [Appendix D: Accessing Dynamics 365 Directly from Rule Helper](#)

Rule Execution Service		
Service Endpoint Id (or Uri)	*	d2e50ca1-ef4c-e611-80e9-6c3be5a82b30
Rule Execution Log Enabled	+	No
Return Full Entity Image	+	No
AppDomain Cache	+	500

Clear Cache

Calling ApplyRules (Auto Fire Mode Rule Sets)

To execute a Rule Set with its Fire Mode set to “Auto,” simply do not define a Rule Set for your rule configuration and associate that rule configuration to the entity type that your auto Rule Set is associated with.

Appendix G: Endpoint Override Configuration

As of version 5.5, InRule for Dynamics now supports Overriding Endpoint Configuration via Azure App Service App Settings. This allows for the overriding of various endpoint settings configured on a rule app, such as REST API URLs or Database Connection Strings, by setting App Settings on your execution service App Service.

To set an endpoint override on your app service, simply navigate to your rule execution app service and go to the Configuration view:

New application setting

Show values

Advanced edit

Filter

Name	Value	Source	Deployment slot setting	Delete
inrule:crm:catalog:label	<div><div></div>Hidden value. Click show values button :</div>	App Config		<div></div>
inrule:crm:catalog:password	<div><div></div>Hidden value. Click show values button :</div>	App Config		<div></div>
inrule:crm:catalog:ruleAppDirectory	<div><div></div>Hidden value. Click show values button :</div>	App Config		<div></div>
inrule:crm:catalog:sso	<div><div></div>Hidden value. Click show values button :</div>	App Config		<div></div>
inrule:crm:catalog:uri	<div><div></div>Hidden value. Click show values button :</div>	App Config		<div></div>
inrule:crm:catalog:useInRuleCatalog	<div><div></div>Hidden value. Click show values button :</div>	App Config		<div></div>
inrule:crm:catalog:user	<div><div></div>Hidden value. Click show values button :</div>	App Config		<div></div>
inrule:crms2s:azureAppId	<div><div></div>Hidden value. Click show values button :</div>	App Config		<div></div>

Select “New Application Setting”

Add/Edit application setting

Name

Value

The name of the override uses the following convention:

inrule:runtime:overrides:<YourEndpointNameHere>:<OverrideType>:<OverrideSetting>

Your endpoint name should match the name of the endpoint or data object in the rule app you wish to override. The available override types are:

AppSettings Key	Description
inrule:runtime:overrides:<endpoint-name>:DatabaseConnection:ConnectionString	(string) Database connection string.
inrule:runtime:overrides:<endpoint-name>:SendMailServer:ServerAddress	(string) Mail server host name.
inrule:runtime:overrides:<endpoint-name>:WebService:WsdlUri	(string) Web service WSDL URI.
inrule:runtime:overrides:<endpoint-name>:WebService:ServiceUriOverride	(string) Web service SOAP end point URI.

inrule:runtime:overrides:<endpoint-name>:WebService:WebServiceMaxReceivedMessageSize	(integer) Web service client max received message size in bytes. (max 2147483647)
inrule:runtime:overrides:<endpoint-name>:XmlDocumentPath:XmlPath	(string) XML document file path on Runtime Service.
inrule:runtime:overrides:<endpoint-name>:XmlSchema:XsdPath	(string) XML schema path or URL.
inrule:runtime:overrides:<endpoint-name>:XmlSchema:EnableXsdValidation	(boolean) Whether to validate XML Entity state against XSD. (true or false)
inrule:runtime:overrides:<endpoint-name>:RestService:RestServiceRootUrl	(string) REST service root URL.
inrule:runtime:overrides:<endpoint-name>:RestService:AuthenticationType	(string) REST service authentication type. (None, Basic, NTLM, Kerberos, Custom)
inrule:runtime:overrides:<endpoint-name>:RestService:RestServiceUserName	(string) REST service authentication username.
inrule:runtime:overrides:<endpoint-name>:RestService:RestServicePassword	(string) REST service authentication password.
inrule:runtime:overrides:<endpoint-name>:RestService:RestServiceDomain	(string) REST service authentication domain.
inrule:runtime:overrides:<endpoint-name>:RestService:RestServiceX509CertificatePath	(string) REST service X509 client certificate path on Runtime Service.
inrule:runtime:overrides:<endpoint-name>:RestService:RestServiceX509CertificatePassword	(string) REST service X509 client certificate password.
inrule:runtime:overrides:<endpoint-name>:RestService:RestServiceAllowUntrustedCertificates	(boolean) Whether to allow REST service certificates not signed by trusted CA. (true or false)
inrule:runtime:overrides:<dataElement-name>:SqlQuery:Query	(string) SQL query text.
inrule:runtime:overrides:<dataElement-name>:InlineTable:TableSettings	(string) XML serialized string of TableSettings object. (see 1.)
inrule:runtime:overrides:<dataElement-name>:InlineValueList:ValueListItems	(string) XML serialized string of ValueListItem collection (see 2.)
inrule:runtime:overrides:<dataElement-name>:InlineXmlDocument:InlineXml	(string) XML document.
inrule:runtime:overrides:<dataelement-name>:RestOperation:UriTemplate	(string) REST operation URI template
inrule:runtime:overrides:<dataelement-name>:RestOperation:Body	(string) REST operation body
inrule:runtime:overrides:<dataelement-name>:RestOperation:Headers:<header-name>	(string) REST operation header

The endpoint setting is the name of the setting to override for that Endpoint Type. Some Endpoint Types have several Endpoint Settings; the available settings for each Endpoint Type can be read about in the [InRule support site documentation](#).

Below is what a properly configured end-result would look like, using a DatabaseConnection override as an example:

Add/Edit application setting

Name	inrule:runtime:overrides:TestDb:DatabaseConnection:ConnectionString
Value	Provider=;Server=;Database=;User Id=;Password=;

The value of the override App Setting would then be set to whatever value you wish to override with. Once your override is set, simply save the changes to your app service. Upon the next execution of rules, the specified endpoint type will be overridden with the supplied value.


Rest Service Override

Unlike other override types, the RestService override type is made of multiple sub-types listed below. When overriding a rest service endpoint, only include the 'RestService' override type in the key, not the sub-type. For example, if overriding the reset service certificate path, you would need to create two app settings with the following keys:

```
inrule:runtime:overrides:YouRestServiceName:RestService:RestServiceX509CertificatePath
inrule:runtime:overrides:YouRestServiceName:RestService:RestServiceX509CertificatePassword
```

The following are the RestService override sub-types and associated settings

- RestServiceRootUrl
 - RestServiceRootUrl
- RestServiceAuthenticationType
 - AuthenticationType
 - RestServiceUserName
 - RestServicePassword
 - RestServiceDomain
- RestServiceX509CertificatePath
 - RestServiceX509CertificatePath
 - RestServiceX509CertificatePassword
- RestServiceAllowUntrustedCertificates
 - RestServiceAllowUntrustedCertificates

 **Important:** If an override type contains multiple settings, like RestServiceX509CertificatePath or RestServiceAuthenticationType, be sure to include an App Setting for each of the settings listed in the documentation.

Appendix H: Azure App Service Plan & Application Insights Configuration

Azure App Service Plan Overview

The Dynamics Rule Execution Azure App Service runs on an Azure App Service Plan. The ARM Template deployment process outlined in [Section 3.3.2: Rule Execution App Service for Dynamics 365](#) will, by default, automatically deploy an App Service Plan for you.

This App Service Plan will be deployed to the subscription and resource group provided during the deployment process. Additionally, the plan will be configured with a “B1” (Basic, Small) pricing tier. This is the **lowest** tier possible for the Rule Execution App Service to run in, as it is the lowest tier that allows running in **Always On** mode. **Always On** is required to permit a continuous web job to run on the app service. Thus, regardless of whether or not you opt to allow the ARM template to create an App Service Plan for you, or use a pre-existing one, the plan must, at a minimum, be of the “B1” tier or higher.

Important: If you leave the ARM template configured to re-deploy the App Service Plan, updating an existing one, the pricing tier of that App Service Plan **will be set to the default pricing tier, regardless of what it may currently be set to**. If you do not wish the pricing tier to be reverted to default, it is recommended you follow the steps below.

Should you wish to use a pre-existing Azure App Service Plan rather than have a new one created for you, a few configuration steps within the ARM template itself are necessary.

Configuring the ARM Template to Use an Existing Azure App Service Plan

1: Locate InRule.Dynamics.Service.parameters.json

The ARM template parameters file is located in the *RuleExecutionAzureService* folder as, defined in [Section 3.3.2: Rule Execution App Service for Dynamics 365](#)

InRule for Microsoft Dynamics CRM Integration Framework > RuleExecutionAzureService				
	Name	Date modified	Type	Size
	azuredeploy.json	9/26/2018 12:49 PM	JSON File	6 KB
	azuredeploy.parameters.json	9/26/2018 12:49 PM	JSON File	1 KB
	InRule.Crm.WebJob.zip	9/26/2018 12:49 PM	Compressed (zipp...	4,974 KB
	Register-AzureApp.ps1	9/26/2018 1:48 PM	PS1 File	0 KB

2: Populate “appServicePlanName” parameter

Open the file in your text editor of choice. First, populate the “**appServicePlanName**” parameter at the bottom of the parameters file. Set the value equal to **the name of your app service plan**.

```
    },  
    "appServicePlanName": {  
      "value": "yourAppServicePlanName"  
    }  
  }  
}
```

3: Update “createOrUpdateAppServicePlan” parameter

Next, you'll need to set the parameter called “createOrUpdateAppServicePlan” to **false**.

```
    },  
    "appServicePlanName": {  
      "value": "yourAppServicePlanName"  
    }  
  },  
  "createOrUpdateAppServicePlan": {  
    "value": false  
  }  
}
```

4: [Optional] Create “servicePlanResourceGroupName” parameter

In the event the App Service Plan you intend to use is located in a different resource group than the one you are deploying the ARM template against, you need to add a parameter to inform the ARM template what resource group your App Service Plan is in. Create the “servicePlanResourceGroupName” parameter as shown below and define the value as **the name of the resource group your App Service Plan exists in**.

```
    },  
    "createOrUpdateAppServicePlan": {  
      "value": false  
    }  
  },  
  "servicePlanResourceGroupName": {  
    "value": "yourResourceGroupName"  
  }  
}
```

5: Save InRule.Dynamics.Service.parameters.json and continue deployment

Save and close the file. You can now proceed with the deployment process outlined in [Section 3.3.2: Rule Execution App Service for Dynamics 365](#) as normal; your rule execution app service will now deploy to the App Service Plan you defined in the steps above

Azure Application Insights Overview

For improved logging capabilities, the Dynamics Rule Execution Service is configured to use an Azure App Insights resource as a logging sink in addition to the logging the App Service itself already has. ARM Template deployment process outlined in [Section 3.3.2: Rule Execution App Service for Dynamics 365](#) will, by default, automatically deploy an App Insights resource for you.

The app insights resource will aggregate all the logs generated from the execution service. These logs can be tremendously useful for debugging any issues encountered with the Rule Execution Service. Depending on the level of event logging configured for the [Rule Execution Service Event Log](#), these logs can add insight into rule executions, entity loading, overall execution timing, and any errors encountered during rule execution.


App Insights in non-Standard Azure Instances (Government Cloud)

If this is your first time deploying the arm template and you would like for the template to create the app insights resource for you, then no other configuration is required.

However, if you would like to use a pre-existing app insights resource then you need to set the **appInsightsInstrumentationKey** and the **appInsightsConnectionString** for that resource in the `azuredeploy.parameters.json`. Then proceed with the rest of the deployment steps outlines in [Section 3.3.2: Rule Execution App Service for Dynamics 365](#)

Appendix I: InRule SaaS Portal Configuration

InRule SaaS offers customers the most streamlined deployment process, eliminating the need to install and manage the Azure App Services, as InRule will manage the deployment of the Rule Execution Service for Dynamics 365 for you. SaaS customers are also able to use their SaaS portal to access the information needed for configuring the InRule Solution for Dynamics 365 and provide Service Account credentials for the SaaS-hosted Rule Execution Service to connect to a Dynamics environment.

 **Important:** Currently only the settings under the 'Dynamics 365 Runtime Settings' configuration section in the portal are applied to the Dynamics execution service. Other sections such as 'Execution Runtime Overrides' are not applied to Dynamics and will need to be managed by InRule support

Granting Consent and Creating Application User

To allow the Rule Execution Service to connect with the Dynamics environment, an administrator for the Dynamics 365 environment will then need to grant consent. For those wishing to use an S2S connection, an application user will then need to be created. This can all be done through the InRule SaaS page provided in the InRule Solution for Dynamics 365.

InRule SaaS - Authentication Setup

InRule SaaS Configuration

This page performs the steps required to enable InRule SaaS authentication. For self-hosted deployments, these configuration steps are not applicable.

Grant Admin Consent	Create Application User
<p>For the InRule Rule Execution App Service to interact with the Dynamics 365 environment, Admin Consent must be granted to the execution service. The button below for opening the admin consent page will direct users to a page where an organization's Azure AD administrator may grant consent.</p> <p>Open Admin Consent Page</p>	<p>S2S authentication requires the creation of a Dynamics 365 Application User associated to the InRule Azure AD Application. Use the button below to create the InRule Application User and assign the necessary role and permissions.</p> <p>Create Application User</p>

Dynamics 365 Runtime Settings in the InRule SaaS Portal

SaaS Customers can find the configuration needed for connecting to their Dynamics 365 instance through the Configuration page of their SaaS portal. Please note that only the Dynamics 365 Runtime Settings are currently used by the Dynamics Execution Service. In a future release, settings from the Execution Server Settings and General Settings section will also contribute to the Dynamics Execution Service. In the interim, if you need any of those settings changed, please reach out to [InRule Support](#).

Configuration

Dynamics 365 Runtime Settings


Execution Runtime Overrides

Execution Server Settings

General Settings

Providing Connection Information

Clicking the drop-down arrow for the Dynamics 365 Runtime Settings section will allow users to provide connection information that will allow the SaaS-hosted Rule Execution Service to interact with their Dynamics environment. You have two options for authentication type, S2S and OAuth. If you choose S2S, InRule will manage the credentials for accessing Dynamics, and you will just need to create the associated Application User in your Dynamics instance. If you want to provide and manage your own credentials, you can choose OAuth, which will authenticate with a named user account.

 **Important:** When changes are made in the SaaS portal it will take 1 minute for the SaaS-hosted Rule Execution Service to refresh and use the updated information.

1: Using an OAuth Connection

For an OAuth connection, the Organization URL as well a [Service Account](#) username and password will need to be provided.

Configuration

Dynamics 365 Runtime Settings

Connection

Show Values

Test Connection

Inherited From	Setting	Value
Tenant: trial-b2v12a-451	Username	(Value Hidden)
Tenant: trial-b2v12a-451	Password	(Value Hidden)
Tenant: trial-b2v12a-451	Url	(Value Hidden)
Tenant: trial-b2v12a-451	Auth Type	OAuth

2: Using an S2S Connection

For an S2S connection, only the Organization URL will need to be configured in the portal.

Configuration

Dynamics 365 Runtime Settings

Connection

Show Values

Test Connection

Inherited From	Setting	Value
Tenant: trial-b2v12a-451	Url	(Value Hidden)
Tenant: trial-b2v12a-451	Auth Type	S2S

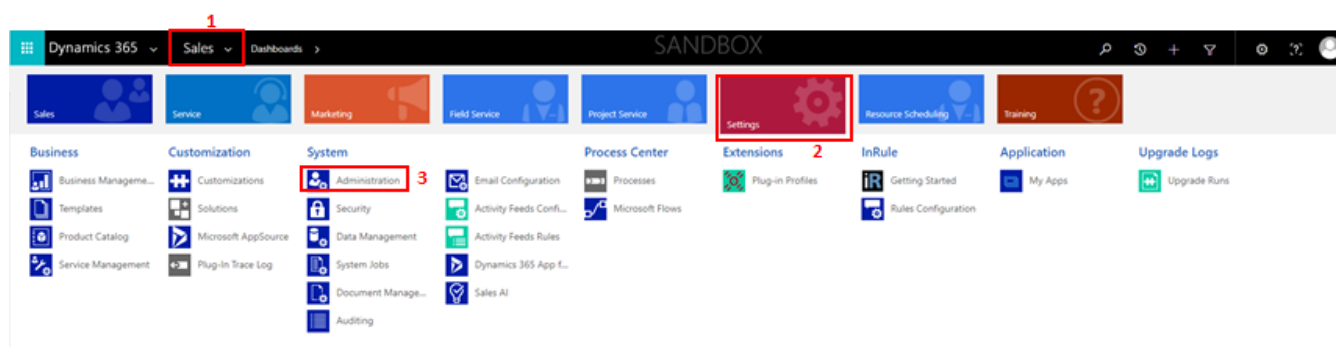
Appendix J: Dynamics 365 Tracing and InRule Event Logging

Dynamics 365 Plug-In Trace Logging

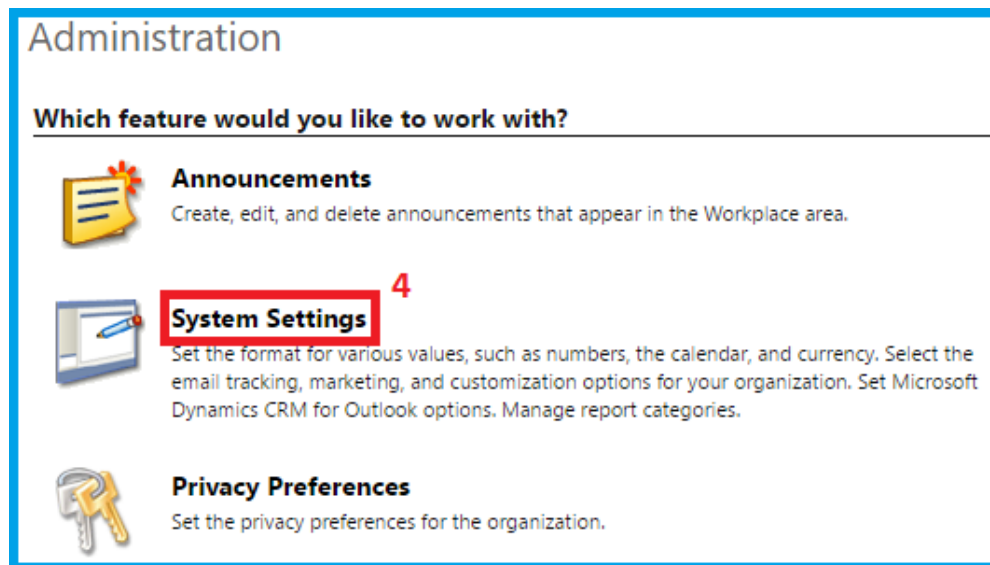
The Dynamics Plug-In Trace Log feature is a debugging tool made available within Dynamics itself for reviewing plugin events and exceptions. This section will highlight how to enable this feature within Dynamics and how to utilize it. Currently, Plug-In Trace Logs only work with the Execution Service in online deployments. For more detail on the limitations of on-prem deployments, reference [Appendix M: Known Issues, Limitations and Troubleshooting](#).

Enable Plug-In Trace Log

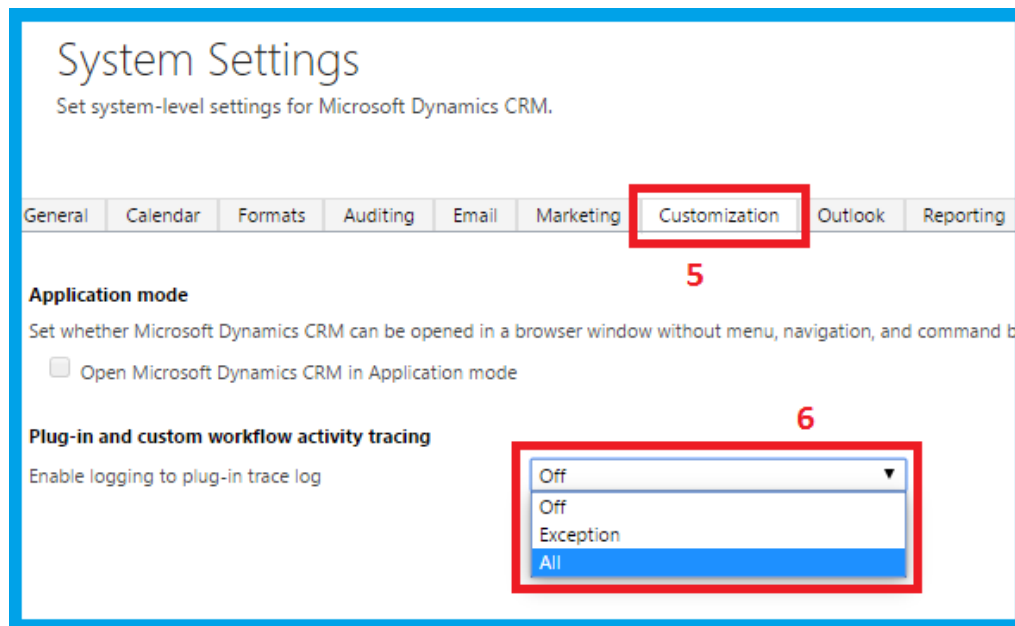
To enable the Plug-in Trace Log, first navigate to **Settings > Administration**.



Once you're in the Administration section, click on **System Settings**.

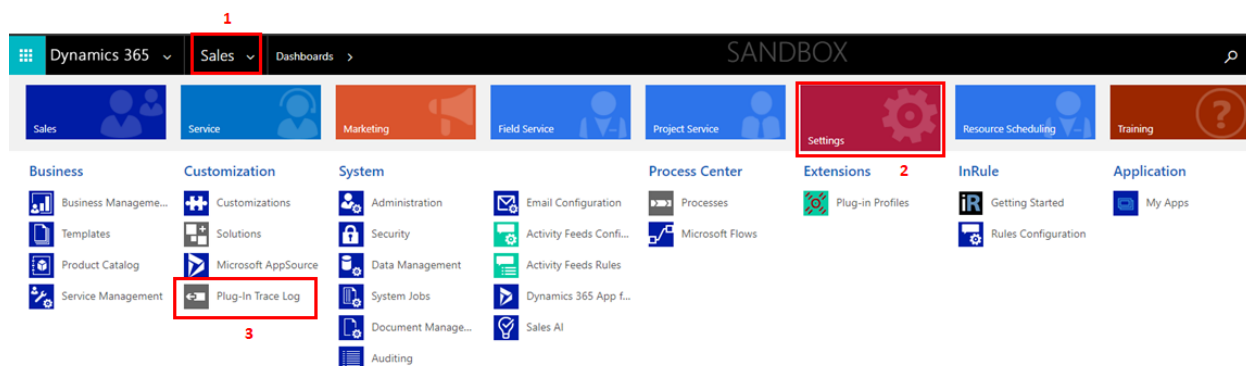


From here you can navigate to the **Customization** tab and select **All** for the **Enable logging to plug-in trace log** field. Then click **OK**. This will create trace logs for all plugin events within your Dynamics environment. In the context of InRule, this includes Create, Update, and Custom Action events.



Viewing Plug-In Trace Log

To review Plug-In Trace Logs after they have been enabled, navigate to **Settings > Plug-in Trace Log**.



Here, there will be a list populated with all logged plugin events. If you do not see a list of logs in a similar fashion as below, that means no plugin events have been logged.

More Actions

<input type="checkbox"/>	System Creat...	Operation Ty...	Type Name	Message Name	Execution Start Time ↓
	Yes	Plug-in	InRule.Crm.ServicePlugins.Onli...	inrule_RulesEngineAction	1/21/2019 12:53 PM
	Yes	Plug-in	InRule.Crm.ServicePlugins.Onli...	inrule_RulesEngineAction	1/21/2019 12:52 PM
	Yes	Plug-in	InRule.Crm.ServicePlugins.Onli...	inrule_RulesEngineAction	1/21/2019 12:52 PM
	Yes	Plug-in	InRule.Crm.ServicePlugins.Onli...	inrule_RulesEngineAction	1/21/2019 12:52 PM
	Yes	Plug-in	InRule.Crm.ServicePlugins.Onli...	inrule_RulesEngineAction	1/21/2019 12:11 PM
	Yes	Plug-in	InRule.Crm.ServicePlugins.Onli...	inrule_RulesEngineAction	1/21/2019 12:10 PM
	Yes	Plug-in	InRule.Crm.ServicePlugins.Onli...	inrule_RulesEngineAction	1/21/2019 12:10 PM
	Yes	Plug-in	InRule.Crm.ServicePlugins.Onli...	inrule_RulesEngineAction	1/21/2019 12:10 PM

For this example, we will view the resulting log of executing a Custom Action from manually running rules. To view a specific log, simply click on the hyperlinked **Type Name** text.

The first section within a specific trace log is the Configuration section, which details the various configuration details about the event, including the Rule Configuration ID used, the event type, the plugin step ID, and more. This section can be useful for debugging by providing a quick means of determining whether or not the rules executed were properly configured.

Configuration

General			
System Created	Yes		
Type Name	InRule.Crm.ServicePlugins.OnlinePlugin		
Message Name	inrule_RulesEngineAction	Primary Entity	none
Configuration	fd95815e-f4b5-e711-8162-e0071b72b791	Secure Configuration	--
Persistence Key	00000000-0000-0000-0000-000000000000	Operation Type	Plug-in
Plugin Step Id	070ad2c4-ae70-e711-811a-e0071b6a6141		
Context			
Depth	1	Mode	Synchronous
Correlation Id	d3b2ce70-4af3-4fd7-a8fb-0a38792ef397	Request Id	4eae987f-1be5-467f-b5f5-cff3942fbae

The second section, the Execution section, is typically the most useful for debugging. It will provide a block of all messages logged by the plugin during its execution, as well as any exception details, if the event resulted in an exception.

Execution Start Time	1/21/2019 1:51 PM	Execution Duration (ms)	26,34:
Message Block	Original Input parameters: Parameter name: entityId, Value: {E1F7327C-9D1A-E911-A82C-000D3A31299D} Parameter name: entityType, Value: contact		
Exception Details	Unhandled exception: Exception type: System.ServiceModel.FaultException`1[Microsoft.Xrm.Sdk.OrganizationServiceFault] Message: An unexpected error occurred from ISV code. (ErrorType = ClientError) Unexpected exception from plug-in (Execute): InRu set to an instance of an object.Detail: <OrganizationServiceFault xmlns="http://www.w3.org/2001/XMLSchema-instance" xmlns="http://schemas.microsoft.com/xrm/201 <ActivityId>2a4757b4-fe60-44a6-97d4-45543c322b93</ActivityId> <ErrorCode>-2147220956</ErrorCode> <ErrorDetails xmlns:d2p1="http://schemas.datacontract.org/2004/07/System.Collections.Generic" /> <Message>An unexpected error occurred from ISV code. (ErrorType = ClientError) Unexpected exception from plug-in (Execute): In not set to an instance of an object.</Message> <Timestamp>2019-01-21T19:51:39.3705065Z</Timestamp> <ExceptionRetriable>>false</ExceptionRetriable> <ExceptionSource>PluginExecution</ExceptionSource> <InnerFault i:nil="true" /> <OriginalException>System.NullReferenceException at InRule.Crm.ServicePlugins.OnlinePlugin.Execute(IServiceProvider serviceProvider) at Microsoft.Crm.Sandbox.SandboxAppDomainHelper.<C__DisplayClass14_3.>Execute(b_00) --- End of stack trace from previous location where exception was thrown --- at System.Runtime.ExceptionServices.ExceptionDispatchInfo.Throw() at Microsoft.Crm.Sandbox.SandboxAppDomainHelper.Execute(IOrganizationServiceFactory organizationServiceFactory, Dictionary`2 pluginSecureConfig, IPluginExecutionContext requestContext, Boolean enablePluginStackTrace, Boolean chaosFailAppDomain) at Microsoft.Crm.Sandbox.SandboxAppDomainHelper.Execute(IOrganizationServiceFactory organizationServiceFactory, Dictionary`2 pluginSecureConfig, IPluginExecutionContext requestContext, Boolean enablePluginStackTrace, Boolean chaosFailAppDomain) at Microsoft.Crm.Sandbox.SandboxWorker.Execute(SandboxCallInfo callInfo, SandboxPluginExecutionContext requestContext, Guid pluginTypeName, String pluginConfiguration, String pluginSecureConfig, SandboxRequestCounter& workerCounter, Boolean r <TraceText> Entered InRule.Crm.ServicePlugins.OnlinePlugin.Execute() Plugin context retrieved. Message is inrule_RulesEngineAction Loading configuration entity with ID 133ca426-9d1a-e911-a82c-000d3a31299d		

The message block logs rule execution “milestones.” In the event that the plugin fails to execute properly, the message block is useful for determining at what point it is failing. The Exception Details block will provide any related details to any exception thrown and is generally the first place to look to diagnose the nature of a plugin execution failure.

Additionally, when interested in performance information, the application logs will also provide insight into data loading time. As can be seen below, this is broken up into total loading time, which is how long the entire loading process takes with internal processing included, total org time, which is strictly the summed total of all entity loading times, and then loading times for each collection loaded. This will also display the total number of entities in each collection.

```
Entity Loading:
Total Loading Time: 233.2697ms
Total Org Service Time: 216ms
Related Entities: account -> systemuser lookup <> (1), account -> contact collection <contact_customer_accounts.> (2), Time: 142.739ms
Related Entities: systemuser -> queue lookup <queue_system_user> (1), Time: 74.5424ms
```

Rule Execution Service Event Log

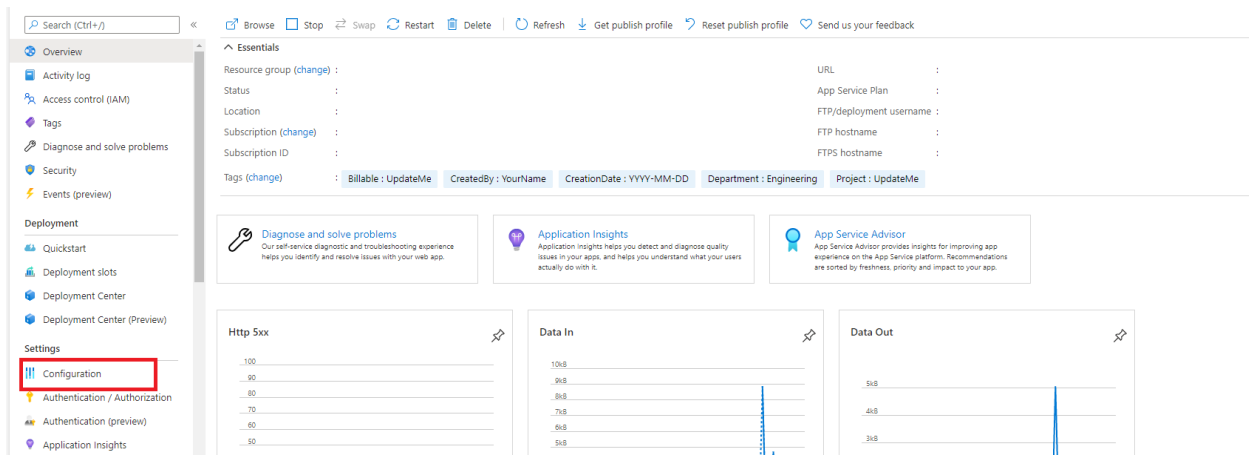
Event logging can be enabled in the Rule Execution App Service to monitor application events. These logs can be tremendously useful for debugging any issues encountered with the Rule Execution Service.

Configuring Event Logging Levels

The Application Event Log can quickly become bogged down with too many logs, making finding specific logs that you may be interested in more difficult. To cut down on excessive informational logs, the Rule Execution Service, by default, will be deployed with a logging level of “Warn,” meaning only Warnings and Errors will be logged. However, this can be adjusted as needed for whatever your needs may be.

To adjust your Rule Execution Service’s logging level, navigate to your App Service as created as a part of the Azure deployment process detailed in [Section 3.3.2: Rule Execution App Service for Dynamics 365](#)

Once you’re looking at the overview of your app service, select **Configuration** in the settings menu:



Scroll down until you see the **Configuration** section:

Application settings

Application settings are encrypted at rest and transmitted over an encrypted channel. You can choose to display them in plain text in your browser by using the controls below. Application Settings are exposed as environment variables for access by your application at runtime. [Learn more](#)

+ New application setting Show values Advanced edit

Filter application settings

Name	Value	Source	Deployment slot setting	Delete	Edit
APPLICATIONINSIGHTS_CONNECTION_STRING	Hidden value. Click to show value	App Config			
inrule:crmcatalog:label	Hidden value. Click to show value	App Config			
inrule:crmcatalog:password	Hidden value. Click to show value	App Config			
inrule:crmcatalog:ruleAppDirectory	Hidden value. Click to show value	App Config			
inrule:crmcatalog:sso	Hidden value. Click to show value	App Config			
inrule:crmcatalog:uri	Hidden value. Click to show value	App Config			
inrule:crmcatalog:useInRuleCatalog	Hidden value. Click to show value	App Config			
inrule:crmcatalog:user	Hidden value. Click to show value	App Config			
inrule:crms2:sazureAppId	Hidden value. Click to show value	App Config			
inrule:crms2:sazureAppSecret	Hidden value. Click to show value	App Config			
inrule:crms2:scrmOrgUrl	Hidden value. Click to show value	App Config			
inrule:crms:serviceBus:key	Hidden value. Click to show value	App Config			
inrule:crms:serviceBus:namespace	Hidden value. Click to show value	App Config			
inrule:crms:serviceBus:owner	Hidden value. Click to show value	App Config			
inrule:crms:serviceBus:path	Hidden value. Click to show value	App Config			
inrule:logging:level	Hidden value. Click to show value	App Config			
inrule:repository:licensing:licenseFolder	Hidden value. Click to show value	App Config			

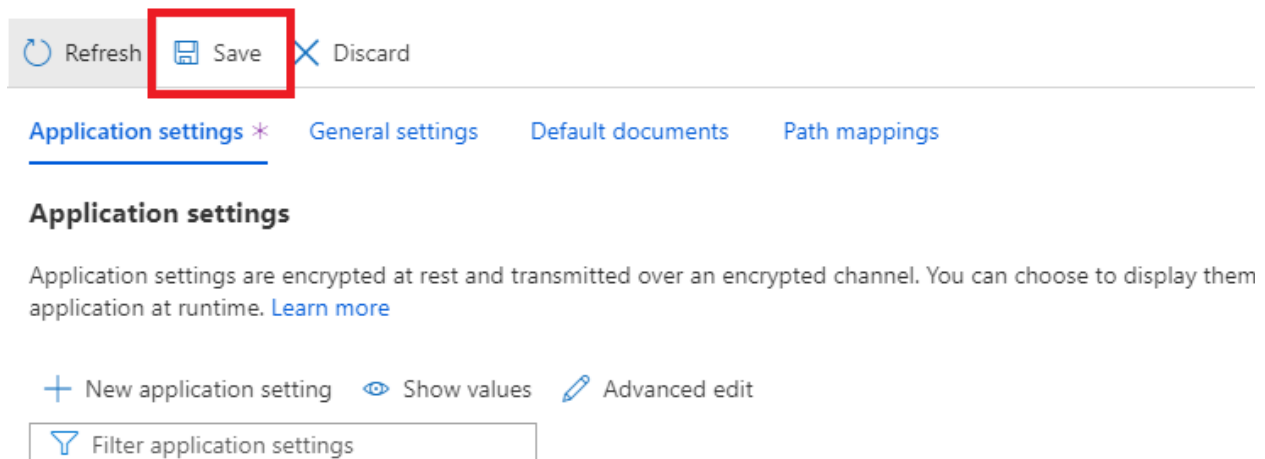
Locate the **inrule:logging:level** setting. Note that its value is currently set to “Warn.”

inrule:logging:level	Warn
----------------------	------

Simply change the value to the desired logging level. You may select from one of the following levels that the Rule Execution Service leverages:

Logging Level	Details
Info	Logs all application events, including Informational events that track the general flow of the application
Warn	Logs Warning and Error events. Warning events highlight abnormal or unexpected events in the application flow, but don't otherwise cause application execution to stop
Error	Logs only Error events. Error events result in the halted execution of the application's current activity due to a failure

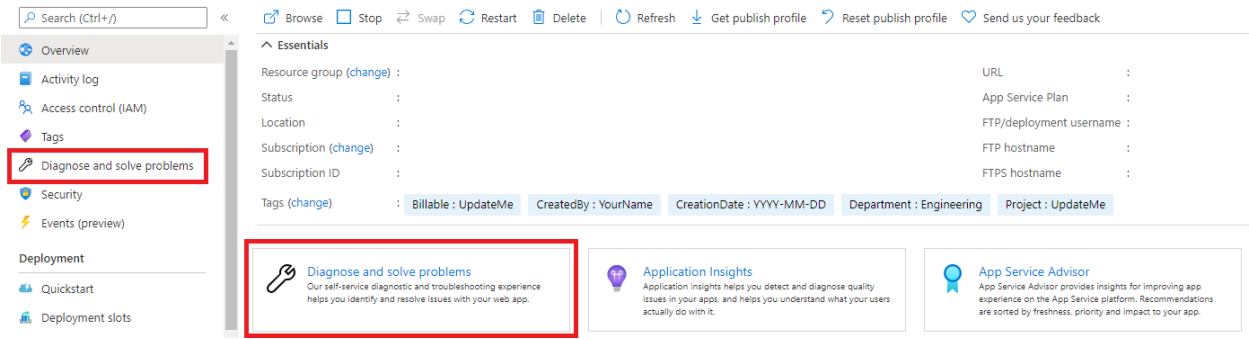
Once you have configured the setting to the desired level, press Save at the top of the page:



Viewing Application Event Logs

To enable event logging, login to Azure and navigate to your App Service as created as a part of the Azure deployment process detailed in [Section 3.3.2: Rule Execution App Service for Dynamics 365](#).

Once you're looking at the overview of your app service, select **Diagnose and solve problems**



Select **Diagnostic Tools**

Troubleshooting categories

Availability and Performance
Check your app's health and discover app or platform issues.
Ex: Downtime, 5xx, 4xx, CPU, Memory
[Troubleshoot](#)

Configuration and Management
Find out if your app service features are misconfigured.
Ex: Backups, Slots, Swaps, Scaling
[Troubleshoot](#)

SSL and Domains
Discover issues with certificates and custom domains.
Ex: 4xx, Permissions, Auth, Binding
[Troubleshoot](#)

Risk Assessments
Analyze your app for optimal performance and configurations.
Ex: Autoscale, AlwaysOn, Density, ARR
[Troubleshoot](#)

Navigator (Preview)
Track changes on your app and its dependencies.
Ex: Change Analysis, SQL, Dependency
[Troubleshoot](#)

Diagnostic Tools
Run language-specific tools for deeper investigation.
Ex: Profiler, Memory Dump, Auto-Heal
[Troubleshoot](#)

Select **Application Events Logs**

Support Tools



Metrics per Instance (Apps)

View metrics for your app



Metrics per Instance (App Service Plan)

View metrics for your App Service Plan



Application Event Logs

View log messages generated by your app to learn more about exceptions and errors



Failed Request Tracing Logs

View detailed information on failed requests to improve site performance or isolate a specific HTTP error



Advanced Application Restart

Restart individual instances or intelligently restart multiple instances of your app by defining time intervals

You should see a list of all application events logged by the rule execution service, denoted with the notification level, timestamp, source, event ID, and web server. Selecting an event log will cause the log details to appear below the list of logs.

Application Event Logs

View Event Logs containing exceptions and errors generated by your app

Filter by all columns

Level	Date	Source	Event Id	Computer
Info	2021-01-25T17:12:51	Application	18185	RD00155D6C1B53
Info	2021-01-25T17:12:44	Application	18185	RD00155D6C1B53
Info	2021-01-25T17:12:31	Application	18185	RD00155D6C1B53
Info	2021-01-25T17:12:29	Application	18185	RD00155D6C1B53
Info	2021-01-25T17:12:14	Application	18185	RD00155D6C1B53
Info	2021-01-25T17:12:02	Application	18185	RD00155D6C1B53
Info	2021-01-25T17:11:44	Application	18185	RD00155D6C1B53
Info	2021-01-25T17:11:32	Application	18185	RD00155D6C1B53

RuleSession.ExecuteRuleSet("\Account:1/RuleHelperRetrievalTests\"): RunningTotalAll report:

FunctionCompileTime: 93.148ms 5
RuleExecutionTime: 4920.909ms 1
BoundStateRefreshTime: 9.716ms 1
ExternalMethodCallTime: 4920.554ms 2
RunningTotalAllTime: 4920.909ms

In the event of rule service issues, error events in this log stream can be useful for debugging purposes. For example, below is an example of an error event log in an instance where the rule service contacting the catalog looking for a rule application that didn't exist:

```
Time: 10/30/2018 7:19:54.872 PM, Source: WcfCatalogProxy, Message:  
Rule application 'DynamicsRules2' does not exist in the catalog.  
Installer Version: 5.2.0.166  
irSDK Version: 5.2.0.166  
HostAppDomainHeapMemoryMB: 18  
, Detail:  
  
Operating System: Microsoft Windows NT 10.0.14393.0  
Processor Count: 1  
CLR Version: 4.0.30319.42000  
CLR Mode: 32-bit  
Thread Culture: 1033  
ThreadID: 15  
ProcessID: 0  
Installer Version: 5.2.0.166  
irSDK Version: 5.2.0.166  
  
Error Information:  
InRule.Repository.Service.InRuleCatalogException: Rule application 'DynamicsRules2' does not exist in  
the catalog.
```

Typically, the response message at the beginning of the log and the Error Information section provide the most pertinent debugging information.

Appendix K: License Management

Licenses are required for both the Catalog Service and the Rule Execution Service. Whether you’re deploying the Online or On-Prem solution will determine the appropriate method for activating your InRule licenses. For Online installations, you will have an Azure license file provided to you by InRule which can be uploaded either through Azure’s App Service Editor tool or via FTP. Walkthroughs of both means are provided below.

For On-Prem solutions, you will need to leverage the InRule License Activation Utility. A walkthrough of this process can be found [here](#).


Uploading a License File


Azure App Service Editor


To upload the InRule license file to your execution service or your catalog service, navigate to the Azure portal and to the appropriate app service. On the left-hand nav-bar, scroll down until you find the App Service Editor option, under the Development Tools header:


Development Tools

 Clone App


 Console

 Advanced Tools

 App Service Editor (Preview)

 Extensions

On the resulting page, press “Go”

 App Service Editor (Preview)

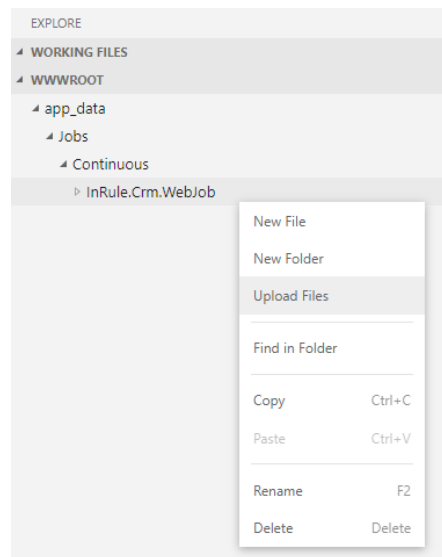
App Service Editor provides an in-browser editing experience for your App code. [Learn more](#)



From here, you’ll need to navigate to the appropriate file location, which is different depending on if you’re adding it for the Execution Service or the Catalog Service:

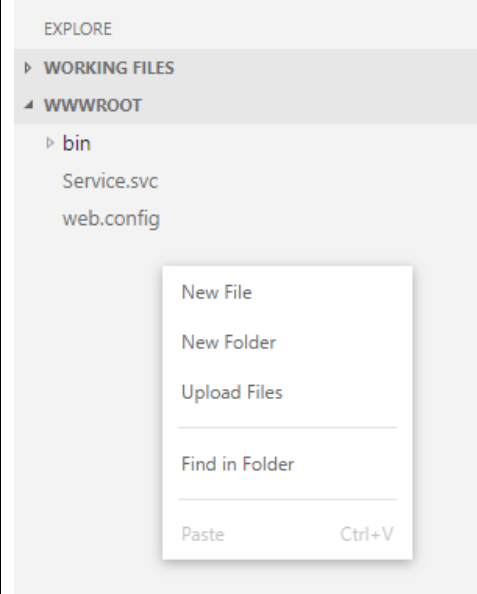
Execution Service	Catalog Service
-------------------	-----------------

Expand the app_data folder and navigate to its inner most directory. Select the InRule.Webjob folder and right-click, then select Upload File:



Find your InRuleLicense.xml file wherever you have it stored and select Open to finish.

Right-click anywhere in the top level of the wwwroot directory, then select Upload File:



Find your InRuleLicense.xml file wherever you have it stored and select Open to finish.

With that, your license file should be uploaded and usable by your Rule Execution or Catalog Service.

FTP

This example leverages Azure CLI in addition to Powershell commands. If you intend to use this method, please run the CLI from Powershell.

Alternative approaches using just Powershell should be possible but are not detailed in this document.

First, retrieve the FTP deployment profile (url and credentials) with the [az webapp deployment list-publishing-profiles](#) command and put the values into a variable:

```
# Example: az webapp deployment list-publishing-profiles --name contoso-execution-prod-wa --resource-group inrule-prod-rg --query "[?contains(publishMethod, 'FTP')].{publishUrl:publishUrl,userName:userName,userPWD:userPWD}[0]" | ConvertFrom-Json -OutVariable creds | Out-Null
```

```
az webapp deployment list-publishing-profiles --name WEB_APP_NAME --resource-group RESOURCE_GROUP_NAME --query "[?contains(publishMethod, 'FTP')].{publishUrl:publishUrl,userName:userName,userPWD:userPWD}[0]" | ConvertFrom-Json -OutVariable creds | Out-Null
```

Then, upload the license file using those retrieved values:

```
# Example:
$client = New-Object System.Net.WebClient;$client.Credentials = New-Object System.Net.NetworkCredential($creds.userName,$creds.userPWD);$uri = New-Object
```

```
System.Uri($creds.publishUrl +  
"/app_data/Jobs/Continuous/InRule.Crm.WebJob/InRuleLicense.xml"); +  
$client.UploadFile($uri, "$pwd\InRuleLicense.xml");  
  
$client = New-Object System.Net.WebClient;$client.Credentials = New-Object  
System.Net.NetworkCredential($creds.userName,$creds.userPWD);$uri = New-Object  
System.Uri($creds.publishUrl +  
"/app_data/Jobs/Continuous/InRule.Crm.WebJob/InRuleLicense.xml"); +  
$client.UploadFile($uri, "LICENSE_FILE_ABSOLUTE_PATH")
```

Activating Your License Utility

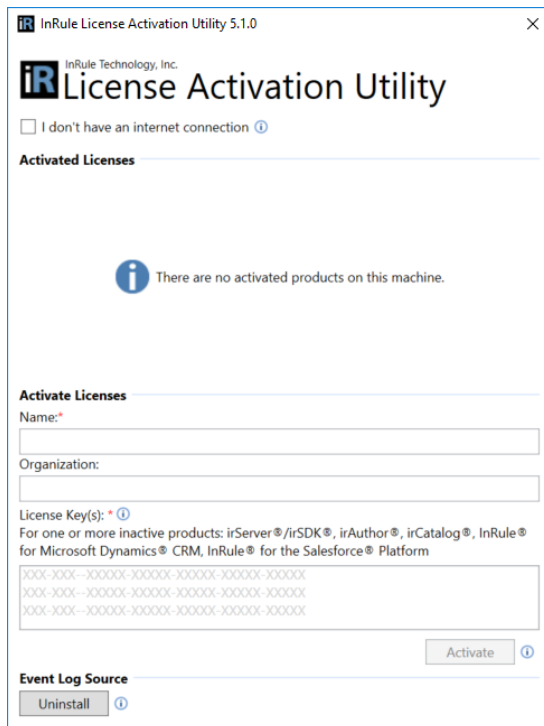
Activating your license for an On-Prem installation of InRule requires leveraging the InRule Activation Utility. A walkthrough of this process can be found below:

1: Download the license activation utility:

Download and install the InRule Activation Utility from support.inrule.com on the server where you intend to deploy the InRule Execution Service.

2: Run the Activation Utility:

Run the activation utility as an Administrator to install Event Log Source.



3: Find your license keys:

Go to support.inrule.com and select Licensing Info on the left-hand navigation bar to find your irServer license keys. Which one(s) you'll need are dependent on what environment you intend to setup your InRule components in.

License Activations

(irServer® Production)

Expiration: [REDACTED]

**** No Activations ****

(irServer® Disaster Recovery)

Expiration: E

*** No Activations ***


§ (b)(7)(C) (irServer® Development)


Expiration: E [REDACTED]


CRM2016-INT (1 cores)	irServer@irSDK® (v5.1.0.150)	7/19/2018	
--------------------------	---------------------------------	-----------	--

4: Enter your name, organization name and license keys into the Activation Utility:


Enter your name, organization and the relevant license keys into the Activation Utility and press Activate

 InRule License Activation Utility 5.1.0




 InRule Technology, Inc.


License Activation Utility

☐ I don't have an internet connection 

Activated Licenses

<input type="checkbox"/>	License Key	Expiration Date
<input type="checkbox"/>	[REDACTED]	
<input type="checkbox"/>	[REDACTED]	

Reactivate 

Deactivate 


Activate Licenses

Name:

Your Name Here

Organization:

Your Org Here


License Key(s): * 

For one or more inactive products: InRule® for the Salesforce® Platform


License Key 1 Here

License Key 2 Here

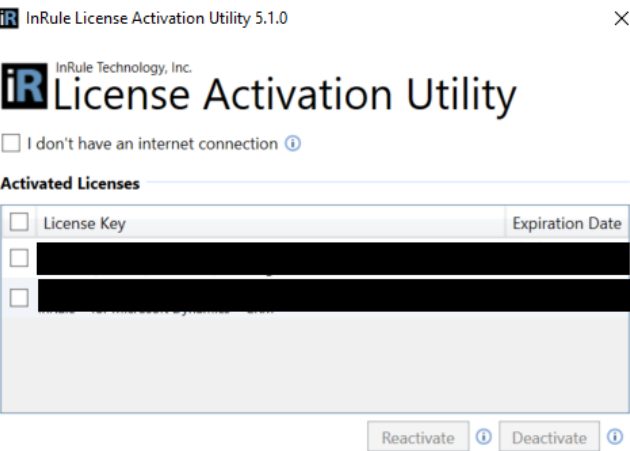
etc|

Activate 

Event Log Source

Uninstall 

5: Verify your license keys have activated:



Appendix L: Redeploying and Upgrading Versions

In most cases, updating InRule for Dynamics is a relatively straight forward process. You will effectively follow the same order of operations of the initial installation steps outlined in this project.

The components that will need to be upgraded are:

- irAuthor
- Azure Rule Execution Service
- InRule for Dynamics Solution

All 3 of these components must be upgraded in all upgrade scenarios.

Beyond that, this appendix discusses some special cases and considerations to be aware of when upgrading.

Version Compatibility Considerations

This list includes specific fixes or changes included in releases. Issue number and title are included here, but you can refer to the release notes section of the support site for further information: [Decision Platform Release Notes – InRule Technology](#). This page includes release notes for all InRule products, but Dynamics release notes will be prefixed with “DYN”



Important: Again, when upgrading to a new version, irAuthor, the InRule for Dynamics Solution, and the Azure Rule Execution Service should always be upgraded together

Issue Number	Version	Description
DYN-442	5.6.0	Added support for advanced authentication to irX, including MFA
DYN-447	5.7.0	Remove solution support for Dynamics 8.2
DYN-445	5.7.0	Deprecate usage of Xrm.Page. As a result, formContext will need to be passed as the first parameter to inRule.executeRules, if calling the function from custom JavaScript
DYN-407	5.7.0	Deprecate custom S2S authentication with built-in connection string support
DYN-502	5.7.0	Rule configurations have been updated to use Rule Set configuration entities rather than a single rule set list field. The new Rule Set configuration allows for setting Display Names, Sort value, Description, and Parameters. This should not affect existing rule configurations; an upgrade for rule configurations to this new model will automatically occur when upgrading to v5.7.0
DYN-558	5.7.2	Added a new page to the InRule solution and sitemap to aid configuration for SaaS setups. This page allows you to grant the required consent to the InRule Azure AD application, and creates the associated Application User for you
DYN-469	5.7.3	Removed legacy S2S authentication settings
DYN-673	5.7.3	Added ruleAppLabel parameter to execution service arm template. By default this label is empty. No ruleAppLabel will result in the most recent version of a rule app being used. Customers relying on a label to denote the production ready version of a rule app, such as the LIVE, will want to add the production label to the arm template prior to upgrade deployment.

SaaS Upgrades

For SaaS customers, the upgrade process will be communicated to you by InRule via email. Keep an eye out for these communications, as they will be the primary source of guidance around what to do in upgrade scenarios. Relevant information from these communications includes when an update will be occurring, the new features coming with the update, the maintenance times required to apply the update, and mandatory steps you'll need to take immediately following the update.

Additionally, as part of the upgrade process, InRule will make a testing environment available for a limited period of time for you to use. This is to allow a window for your organization to perform regression testing to verify that you don't experience any new issues with the new version. The upgrade communication will provide details around when the test environment is available, how to access it, and how long it will be active.

Execution Service App Settings

When running an ARM template deployment to upgrade an existing app service, the new deployment will remove all app settings that currently exist and replace them with the settings during the new deployment. Because of this, it is important to save your parameters file when you do a deployment. If you do not have your previous parameters file you can find it in the deployments section of the Azure Portal. This also means that any settings that were changed or added manually, such as logging level or endpoint overrides, will be removed.

Alternatively, current app setting values can be manually set in the ARM template parameter file prior to deployment to have the template deploy using those values. Should you go this route, be thorough during the transfer process, it is common for values to have been manually changed over time. For a thorough breakdown of the relevant parameters, reference [Section 3.3.2: Rule Execution App Service for Dynamics 365](#)

Lastly, redeploying or upgrading via the ARM template will remove your InRule license from the resulting app service. The license file will need to be manually added back; for a walkthrough on how to do this, reference [Appendix K: License Management](#).

WS-Trust/Office365 Auth Deprecation

As of April 2022, Microsoft has deprecated WS-Trust/Office365 Authentication. If you are intending to leverage S2S authentication, it is strongly recommended you familiarize yourself with Microsoft's announcement and understand the transition to OAuth based connection strings, as any connection strings used will need to be OAuth. You can find Microsoft's documentation on the topic [here](#).

Early versions of irX (<5.6) and the integration framework (<5.2) can only leverage WS-Trust authentication and will not be compatible with this change. For full functional coverage, customers will need to be running these versions or newer to ensure compatibility with this transition.

S2S Legacy Settings Removal

In previous versions of the Rule Execution Service, S2S connection information was provided in individual app settings. Configuring the connection information this way was deprecated in 5.7.0, and support for it has been removed entirely in 5.7.3. The legacy app settings that have been removed are:

- orgUrl
- appld

- appSecret

S2S connection information should now be provided as a connection string in the **crmConnectionString** app setting. See link for configuring connection string information: <https://docs.microsoft.com/en-us/powerapps/developer/common-data-service/xrm-tooling/use-connection-strings-xrm-tooling-connect>

Execution Service App Settings

When running an ARM template deployment to upgrade an existing app service, the new deployment will remove all app settings that currently exist and replace them with the settings during the new deployment. Because of this, it is important to save your parameters file when you do a deployment. If you do not have your previous parameters file, you can find it in the deployments section of the Azure Portal. This also means that any settings that were changed or added manually, such as logging level or endpoint overrides, will be removed.

Alternatively, current app setting values can be manually set in the ARM template parameter file prior to deployment to have the template deploy using those values. Should you go this route, be thorough during the transfer process, it is common for values to have been manually changed over time. For a thorough breakdown of the relevant parameters, reference [Section 3.3.2: Rule Execution App Service for Dynamics 365](#)

Lastly, redeploying or upgrading via the ARM template will remove your InRule license from the resulting app service. The license file will need to be manually added back; for a walkthrough on how to do this, reference [Appendix K: License Management](#).

Solution Customizations

If you have used the Plugin Registration tool to change the 'Run in User's Context' setting on the 'InRule Custom Action Step', you will need to set this back when updating. Additionally, in the unlikely event you have made any other customizations to resources included in the solution, such as the JavaScript web resources, you will need to re-apply these customizations after updating the solution.

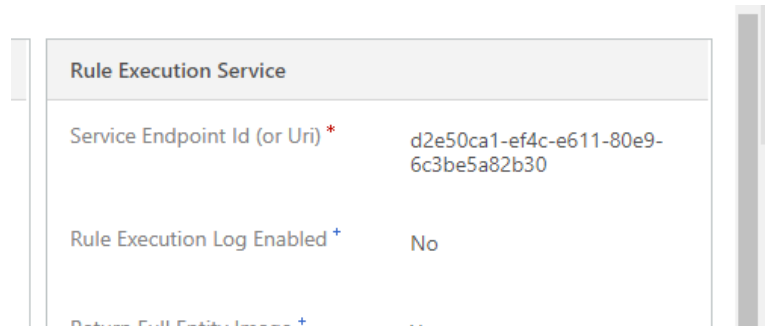
Upgrading from Cloud Service-based versions

Earlier versions of InRule for Dynamics used Azure's legacy Cloud Service platform for the rule execution service. The last version to support this was 5.1.1. Versions from 5.2.0 and on now use Azure App Service for the same purpose. When upgrading from one of these versions, simply follow the steps for creating, deploying, and configuring the new Azure resources. Both the App Service and Cloud Service versions use Azure Relay for communication with Dynamics, but in most cases, it is simpler to let the provided ARM template provision a new Relay for use with the App Service. Continue following the rest of the steps in the Deployment Guide to deploy the new version of the Dynamics package and configure it to point to the new Relay and execution service. Once you have verified the new setup is working, you can delete the old Cloud Service resources.

Switching from Dynamics On-Prem to Online

While there are no universal steps for transitioning from an on-prem to online version of Dynamics, there are a few conceptual differences in the way InRule for Dynamics works in each that you should be aware of. In on-prem installations, communication with the rule execution service is typically handled directly via HTTP. In Dynamics Online, communication is handled via an Azure Relay resource that manages the WCF Relay. Depending on the specifics of how your migration takes place, you may need to re-install the

Dynamics package entirely, or it may still be installed on the migrated instance, along with associated config records. Either way you will need to follow all the steps in [Performing the Installation: In Azure](#) for deploying to Azure, and ensuring that the required Relay information is updated in Dynamics. Once you complete all these steps, you will need to update any existing InRule Configuration records to replace the URI that was in the 'Service Endpoint Id (or Uri)' field with the appropriate value (d2e50ca1-ef4c-e611-80e9-6c3be5a82b30).



Using the new Entity-based Configuration

From version 5.0.28 and on, step registrations can be configured from within Dynamics. This new system uses Rules Configuration records and associates them to step registrations created in the Rules Configuration UI. Before you can use this new configuration, you will need to delete any custom registrations (registrations other than 'InRule Custom Action Step') under the OnlinePlugin:

- ▲ (Assembly) InRule.Crm.ServicePlugins
 - ▲ (Plugin) InRule.Crm.ServicePlugins.OnlinePlugin
 - (Step) InRule Custom Action Step
 - (Step) InRule.Crm.ServicePlugins.OnlinePlugin: Update of contact
 - (Workflow Activity) InRule.Crm.ServicePlugins.RulesEngineActionInvoker

Once you've deleted these registrations, you can create new ones from Dynamics using the steps in [Appendix F: Rules Configuration and Settings](#)

Default Rule App Label Change

As of v5.7.3, the default Rule App Label configured on the execution service has been changed from "LIVE" to not being set with a value. If a default label is not configured on the execution service, it will now default to using the most recent version of the rule app being run. This has the potential to create an issue during upgrade scenarios where users are relying on the LIVE label to denote their production-ready rule. If using the most recent version of the given rule app is not the desired outcome, the default label on the execution service will need to be set back to LIVE (or whatever the appropriate production label for your existing rule apps may be). This can be done in the ARM template prior to upgrade deployment.

Appendix M: Known Issues, Limitations and Troubleshooting

This portion of the document lists current known issues and limitations that may be encountered in usage of the Integration Framework. Most should only be encountered in limited edge cases.

Configuration

As highlighted in [Configure the InRule Solution in Dynamics](#), the 'Test' button on the configuration page can be used to validate that everything is configured correctly for the end-to-end connectivity required to execute rules. The 'Test' button validates three areas which are described in more detail below

Execution Service Connectivity

The first step in the test is validating connectivity to the Rule Execution Service via the Azure Relay. Errors here could indicate a variety of issues, such as misconfigured relay key or address, or network connectivity issues between Dynamics or the execution service and the relay. Double-check the relay config values in both the configuration page in Dynamics and the app settings on the execution service site.

An error here can also be the result of a critical error within the execution service. If a response is returned from the execution service but it is an error, the validation message will be prefixed with 'Error returned from rule service', and the exception message. Further exception details can usually be found in the Application Event Log of the execution service server.

Dynamics Connectivity

Once the test request reaches the execution service, it will attempt to validate the connectivity from the execution service back to Dynamics. Errors here are usually a result of issues in the connection string. This can include an incorrect Dynamics instance URL, incorrect credentials, or a malformed connection string. Even if the credentials are correct there can be issues with the service account used, such as the account not having the correct permissions or licensing, or using username/password credentials for an account that requires MFA. Failures here can also be due to network restrictions between the execution service and the Dynamics API.

Catalog Connectivity

The final step is checking connectivity from the execution service to the rule catalog service. Errors here are usually a result of either an incorrect catalog URL or catalog credentials, but errors in the catalog service can also cause a failure here. A common issue, for example, is an invalid or expired license key on the catalog service.

Execution Runtime

Execution Service 1-Minute Timeout

Because of Dynamics default configuration, a 1-Minute Timeout may occur when a request to the Azure Relay takes longer than a minute to respond. This can occur because of latency between Dynamics and the rule execution service, or anything else that causes the rule execution service to take more than 1 minute to respond.

Plugin Execution 2-Minute Timeout

Dynamics plugin execution is configured to timeout when the plugin runs for more than two minutes. In online implementations there is nothing that can be done about this behavior because the plugin cannot be taken out of sandbox mode. Since On-Prem implementations must be run outside of sandbox mode, this timeout period can be changed by updating the configuration.

Missing rulesets on Run Rules drop-down

If you have users who are missing rule sets on the Run Rules drop-down, verify that the user's role is set to "InRule Rule Executor". Running rules from the dynamics UI requires access to permissions not normally given to most users – the "InRule Rule Executor" role provides the minimum level of permissions needed to execute rules. See [Getting Users Ready](#) for more information.

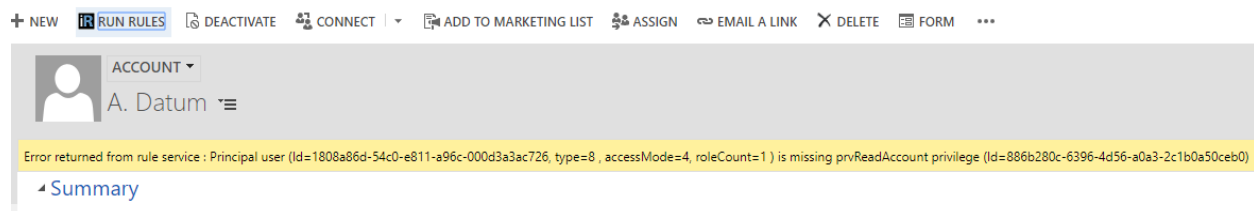
Roles, Privileges and Access

S2S User Settings

If the 'InRule Integration' user is being used with the associated 'InRule Integration Administrator' role it will have read privileges for all entities in the system when first installed, but the entity permissions are not updated after the initial creation. If a new entity type is created, it will not automatically be added to this security role and will have to be added manually. This user account can be assigned to the 'System Administrator' role to alleviate the stale permission issue, as it is automatically updated with permission when new entities are created

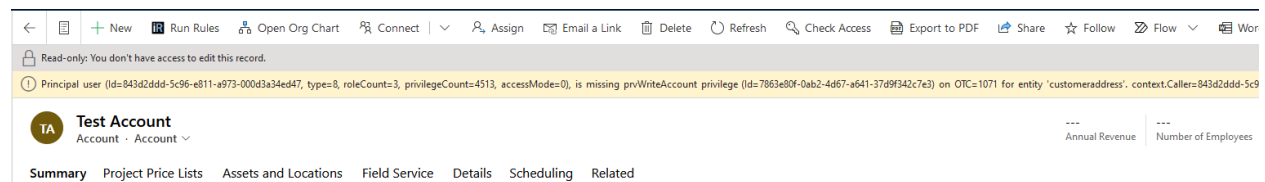
Missing Entity Privilege Error

If, after attempting to run rules, you see an error like the one below, make sure that the 'InRule Integration' user has been assigned the 'InRule Integration Administrator' role, and that the role has the permissions needed to access any entities necessary.



User Principal Permission Error

If your rule updates entity records, you may see an error like the one below when running rules. This is because by default any changes are saved using the current user's account. If the rule runner does not have access to any of the entities being changed, execution will fail with this error.



This error can be resolved in two ways. First, the user's permissions in Dynamics can be updated so that they have the required permissions. Second, the InRule Custom Action step registration can be updated with the plugin registration tool to run under a different user context. By default, this is set to 'Calling User', but if your rules need to update an entity that end users do not have access to, you can change it to 'System', or choose a specific user account that has access.

Update Existing Step

General Configuration Information

Message: inrule_RulesEngineAction

Primary Entity: none

Secondary Entity: none

Filtering Attributes: Message does not support Filtered Attributes

Event Handler: (Plugin) InRule.Crm.ServicePlugins.OnlinePlugin

Step Name: InRule Custom Action Step

Run in User's Context: Calling User

Execution Order: 1

Description: InRule.Crm.ServicePlugins.OnlinePlugin: inrule_RulesEngineAction

Event Pipeline Stage of Execution: PostOperation

Execution Mode: ☐ Asynchronous ☒ Synchronous

Deployment: ☒ Server ☐ Offline

☐ Delete AsyncOperation if StatusCode = Successful

Unsecure Configuration

559ef8c6-9923-eb11-a813-000d3a5a7ad8;account=d905724b-0c26-eb11-a813-000d3a5a7ad8

Secure Configuration

Update Step Close

Using irX for Dynamics with Multi Factor Authentication (MFA)

Advanced login options have been added in irX for Dynamics to allow users with MFA to authenticate. To authenticate in this manner select **Use Advanced Login** in the environment picker. This will reveal a **Login to Dynamics 365 Environment** button. Once selected, the advanced login form will be displayed. Select **Office 365**, enter your credentials, and complete the login process.

Usability

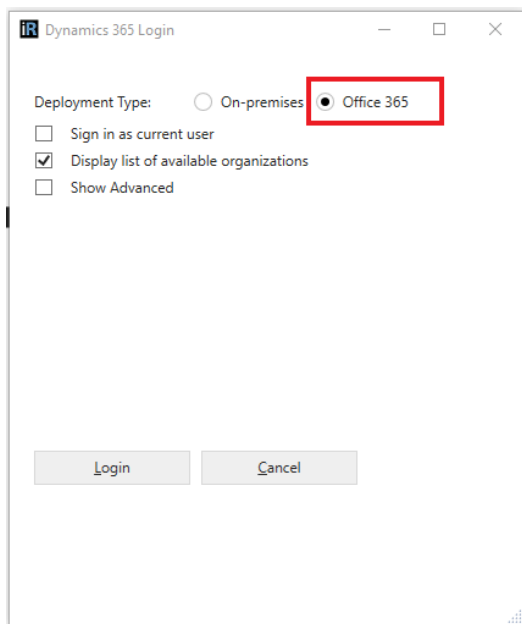
Updating Entity Status via Rules

Currently, irX supports the updating of an entity's status through rules for "standard" entities that can only swap states between "Active" and "Inactive," as well as for the Case entity. Given technical limitations around how Cases are set to the "Resolved" status through rules, doing so currently auto-defines the Case Resolution field as "Resolved by rules." This can be edited within Dynamics itself after the fact if you wish to change it, but there is no way to set the Case Resolution to anything different via rules at this time. Additionally, rules that update an entity from "Resolved" or "Inactive" back to any variety of "Active" that also update another field on the entity are not supported at this time. To support this operation, you will first need to reactivate the entity, then make any desired changes to any fields on that entity after the fact, rather than doing both in a single rule.

Other entities with similar "Resolved" states such as Order are not supported at this time.

On-Prem Execution Mode

Running the plugin in an On-Prem Dynamics environment requires running outside of Sandbox mode. Plugins in sandbox isolation mode run under partial trust, which prevents plugins from doing things like accessing the file system and registry. This also prevents reflection from being used, which is necessary for serializing Dynamics classes for communication over WCF. When using service endpoints to communicate with Azure Relay, Dynamics provides helper classes that handle this serialization, but no such classes are provided for On-Prem communication. It is worth noting that a side-effect of running outside sandbox mode is Dynamics will not write to the plugin trace-log.



Considerations with Advanced Login:

1. Sign in as Current User (SSO) for Office 365 is not currently supported

2. If On-Premises connection is attempted with an Office 365 environment, you may see the following error in the Error Log **A Dataverse server name is required**. To resolve, use Office 365 login.

Using Dynamics Connections

Support for Dynamics Connections is currently limited. You can bring in the Connection entity relationship through irX as well as write rules against the Connection entity, however, the ability to write rules against the Connected entity itself through the Connection relationship is not supported. For example, if you have an Account Connected to a Contact, you can write rules against the Connection entity itself and do things like change the Role, Role Type, or other fields on the Connection entity, however, you will not be able to write rules against the Contact entity. The Connection entity sits between the two Connected entities (in this case Account and Contact) and contains all the Connection metadata describing the Connection.

Locate SAS Key and Namespace Address in Azure Portal

When setting up the connection to the rule execution service in Dynamics, you will need to provide the SAS key and Service Bus Namespace Address for the Azure Relay that the execution service is listening to. If you deploy the Azure resources using the provided ARM template and instructions, you should get this value when the deployment completes.

If you need to locate the SAS key, you can retrieve it by navigating to the relay resource in the Azure portal. The execution service uses the 'RootManageSharedAccessKey' by default, which you can find in the 'Shared access policies' menu on the sidebar. Once you open the policy, you can copy out either the primary or secondary key.

Finally, the Service Bus Namespace Address can be derived from the Name of the Azure Relay or the 'Primary Connection String'. For example, an Azure Relay defined as:

- Name: 123dynamicsrelay
- Connection String (beginning with): Endpoint=sb://123dynamicsrelay.servicebus.windows.net...

The equivalent Service Bus Namespace Address is:

- Azure SB Namespace Address: https://123dynamicsrelay.servicebus.windows.net/ruleexecution

Performance

Plugin Trace Log

When troubleshooting performance issues, the plugin trace log provides detailed metrics to help analyze the various components of rule execution performance. At a high level, the plugin trace logs provide run times for the various execution steps. The 'Performance' section of the plugin trace form includes a field for 'Execution Duration', which is the total process time for the plugin request. In addition, the InRule plugin also provides a more detailed breakdown of steps in the trace text. These are broken into Plugin Setup, Execution Service and Save.

Plugin setup time is usually short and includes config entity loading and parsing of input parameters. Execution Service time will typically be longer, as this includes communication latency over Azure Relay,

loading of additional entity data, and execution of rules. Save time will not always show up in trace logs, but if any entity changes come back from rule execution the time needed to save them will be reflected here. The trace log will also include a message noting the number of entities returned from the execution service for saving. This saving should not take much time for a handful of entities but can take several seconds for large change sets.

For a further breakdown of the data loading and execution time from the rule execution service, the 'Info' level event logs for the App Service can be accessed through the Azure Portal. More information about these logs can be found [below](#).

Included below is a sample plugin trace log.

Execution

Performance			
Execution Start Time	9/21/2020 3:55 AM	Execution Duration (m)	796
Message Block	Entered InRule.Crm.ServicePlugins.OnlinePlugin.Execute() Plugin context retrieved. Message is InRule.RulesEngineAction		
Exception Details	--		

```
Entered InRule.Crm.ServicePlugins.OnlinePlugin.Execute()
Plugin context retrieved. Message is InRule.RulesEngineAction
Loading configuration entity with ID 42260319-bd49-e911-a973-000d3a1d5285
Configured with depth check of: 0
Plugin context depth: 1
Primary Entity name: none
Primary Entity ID: 00000000-0000-0000-0000-000000000000
Step ID: 070ad2c4-ae70-e711-811a-e0071b6a6141
Step Name: InRule Custom Action Step
Original Input parameters:
Parameter name: entityId, Value: f908cdfa-9b94-ea11-a811-000d3a8df742
Parameter name: ruleAppName, Value: AccountRuleApp
Parameter name: ruleSetName, Value: InvoiceReconciliationRules
Parameter name: entityType, Value: account
Parameter name: persistChanges, Value: True
Parameter name: dirtyEntityImage, Value:
Output parameters:
Parameter name: executionResult, Value:
No entity image present in message. Locating and retrieving entity from CRM using ID f908cdfa-9b94-ea11-a811-000d3a8df742 and type account
Posting the execution context to the service endpoint d2e50ca1-ef4c-e611-80e9-6c3be5a82b30 using RuleApp 'AccountRuleApp' and RuleSet 'InvoiceReconciliationRules'.
Plugin Setup Time: 16.6833ms
Plugin Execution Service Time: 704.7022ms
```

```

Received string response from rule execution service. Response length: 2734
Response json data:
{"EntityImage":null,"EntityChangesResponse":{"EntityChanges":[{"Id":"0a38ba18-9c94-ea11-a811-000d3a8df742","ChangeType":0,"Entity":null,"EntityImageContainer":{"Attributes":[{"Name":"totaldaysnonbillable","Value":2.5625}], "EntityName":"account","Id":"0a38ba18-9c94-ea11-a811-000d3a8df742"},"RelationshipContainer":null}]},"NotificationResponse":{"Notifications":[{"Type":0,"Message":"Days Booked: 0"}, {"Type":0,"Message":"Days Used: 0"}, {"Type":0,"Message":"Days Remaining: 0.0000"}, {"Type":0,"Message":"Days Invoiced_Paid: 0"}, {"Type":0,"Message":"Days Invoiced_Open: 0"}, {"Type":0,"Message":"Days Invoiced_Total: 0"}, {"Type":0,"Message":"Amount Invoiced_Total: 0"}, {"Type":0,"Message":"Training Booked: 1.5"}, {"Type":0,"Message":"Training Used: 1.5"}, {"Type":0,"Message":"Total Days Booked: 1.5"}, {"Type":0,"Message":"Total Days Used: 1.5"}, {"Type":0,"Message":"Total Days Comp: 1.5"}, {"Type":0,"Message":"Total Days Non-Billable: 2.5625"}, {"Type":0,"Message":"Services Non-Billable: 0.0313"}, {"Type":0,"Message":"Training Non-Billable: 0.5938"}]},"ValidationResponse":{"Validations":[]},"ContextResponse":{"PropertyBag":[]},"ErrorResponse":{"Errors":[]},"RuleExecutionLog":null}
SAVE: Saving 1 changes to CRM
Processing change -- Update on entity ID 0a38ba18-9c94-ea11-a811-000d3a8df742 -- type account
Synchronizing change images:
Completed sync changes image
Save Time: 74.1592ms
Output parameter "executionResult" populated with execution service response
Exiting InRule.Crm.ServicePlugins.OnlinePlugin.Execute()

```

Plugin Persistence Performance

When changes are made to entities as part of rule execution, these changes are bundled up and sent from the execution service back to the plugin for saving. Performance testing in this area indicates you can expect to be able to save around 10 changes a second, although other things can increase this time, such as other plugins registered to the entity being updated, and initial plugin start-up times. This performance constraint is specific to Dynamics overall (as compared to InRule) and has been verified with direct testing using XRM with simple entities.

Execution Service Performance

Enabling Info level logging on the Execution Service can provide more detailed performance data. The plugin trace log will include some summary data about the entire rule execution run, but the execution service can log specifics about entity data loading and rule execution from the service. To enable the info level logging set the “inrule:logging:level” app setting to “Info”. For all installations this can be done by editing the web.config, but Azure-hosted services can also edit this setting via the Azure portal.

Setting this to “Info” will enable two sets of logs, the standard InRule SDK logs that appear for all InRule products, and the data loading logs that measure entity loading specific to the Dynamics integration. You can find all the details about the Event Logs on the InRule support site [here](#). There are a variety of different events, but the ‘ExecuteRuleSet’ entry will contain some of the most important measurements, such as compile, execution and external call times.

However, these measurements only capture actual rule execution, and before rule execution happens the entity tree is loaded from Dynamics. The data loading log captures this information. When executing rules from Dynamics, only the root entity is sent from the plugin to the execution service, and the execution service will then use the rule app metadata to determine which relationships and entities need to be loaded. The execution service batches these requests for greater efficiency, but very large entity trees or network latency can still lead to long load times. The entity loading log breaks down the total loading time

into each 'batch' of loading requests made, and lists the total load time and all entities loaded in the batch. This information can be used to identify where large numbers of entities are being loaded and how long particular batches take to retrieve from Dynamics

Taken together, these logs can be used with the plugin trace log to get an end-to-end picture of performance, and help identify which steps consume the most time. An extract of a sample of the data loading log is shown below

```
Entity Loading:
Total Loading Time: 911.5114ms
Total Org Service Time: 701ms
Root Entity: inr_projectprofile

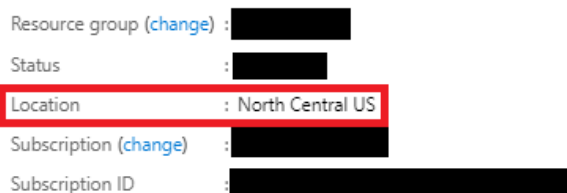
Related Entities: inr_projectprofile -> account lookup <inr_account_inr_projectprofile> (1),
inr_projectprofile -> inr_agreement lookup
<inr_inr_agreement_inr_projectprofile_PrimaryAgreement> (1), inr_projectprofile ->
inr_projectprofilegroup lookup <inr_inr_projectprofilegroup_inr_profile> (1),
inr_projectprofile -> inr_agreement collection <inr_inr_projectprofile_inr_agreement> (2),
inr_projectprofile -> annotation collection <inr_projectprofile_Annotations> (33),
inr_projectprofile -> inr_businessapplicationtype collection
<inr_inr_businessapplicationtype_inr_projectpr> (1), inr_projectprofile ->
inr_integrationdetail collection <inr_inr_integrationdetail_inr_projectprofile> (5),
inr_projectprofile -> inr_runtimeapplicationtype collection
<inr_inr_runtimeapplicationtype_inr_projectpro> (4), Time: 180.1637ms
Related Entities: account -> incident collection <incident_customer_accounts> (43),
inr_agreement -> serviceappointment collection <inr_agreement_ServiceAppointments> (69), Time:
128.366ms
Related Entities: incident -> subject lookup <subject_incidents> (8), serviceappointment ->
service lookup <service_service_appointments> (5), Time: 109.0635ms
```

In addition to always being logged on the execution service when Info logging is enabled, this log can also be returned in the response and logged to the plugin trace log for easier access. This can be enabled by following the steps in [Displaying Additional Logs and Response Data in Trace Logs](#)

Regional Performance

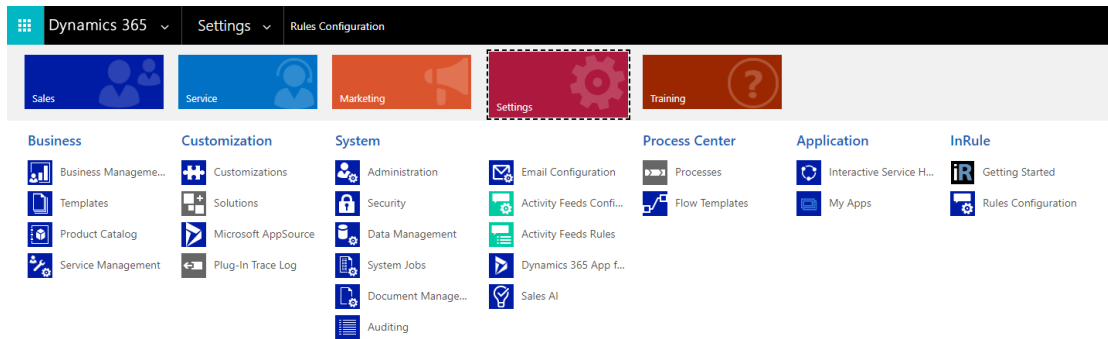
To maximize performance by reducing network traversal time, ensure that your Dynamics 365 instance and Azure resources (App Service, Azure Relay, etc.) are deployed to the same Azure Region.

To determine what region your Azure resources are deployed to, navigate to that resource's overview page in the Azure portal and reference the "Location" property at the top of the page:

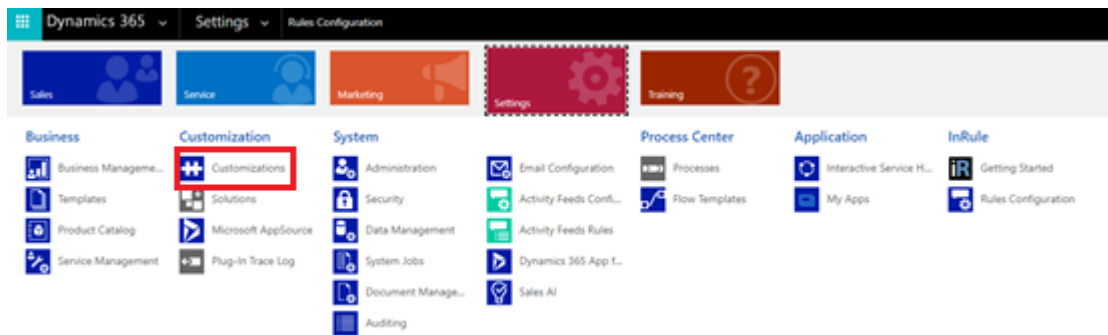


```
Resource group (change) : [REDACTED]
Status : [REDACTED]
Location : North Central US
Subscription (change) : [REDACTED]
Subscription ID : [REDACTED]
```

Determining your Dynamics 365 region is unfortunately less straightforward. First, log in to your Dynamics instance and select Settings:



Under the Customization header, select “Customizations”



Select “Developer Resources”

Customization



Look for the endpoint address in the Discovery Service section:

Developer Resources

Getting Started

[Developer Center](#) [Developer Forums](#) [SDK NuGet Packages](#)
[SDK Download](#) [Sample Code](#) [Developer Overview](#)

Connect your apps to this instance of Dynamics 365

Instance Web API
HTTP REST API providing access to this instance of Dynamics 365. For more information see [Microsoft Dynamics 365 Web API](#).
Service Root URL: <https://rqsandbox2.api.crm.dynamics.com/api/data/v9.1/>

[Download OData Metadata](#)

Instance Reference Information
Use this information to uniquely identify this instance of Dynamics 365. You can use this to retrieve the current URL for this instance. For more information see [Azure extensions for Microsoft Dynamics 365](#).
ID: [5f7789d5-e4a-49c3-bacc-d8d736a9973](#)
Unique Name: [orga496191](#)

Connect your apps to the Dynamics 365 Discovery Service

Discovery Web API
HTTP REST API providing connection information for the set of Dynamics 365 instances to which the caller has access. For more information see [Discover the URL for your organization with Discovery Web API](#).
Endpoint Address: <https://disco.crm.dynamics.com/api/discovery/v9.1/>

[Download OData Metadata](#)

Organization Service
SOAP Service providing access to this instance of Dynamics 365. For more information see [Use the /OrganizationService web service to read and write data or metadata](#).
Endpoint Address: <https://rqsandbox2.api.crm.dynamics.com/XRMServices/2011/Organization.p>

[Download WSDL](#)

Discovery Service
SOAP Service providing connection information for the set of Dynamics 365 instances to which the caller has access. For more information see [Discover the URL for your organization with /DiscoveryService web service](#).
Endpoint Address: <https://disco.crm.dynamics.com/XRMServices/2011/Discovery.svc>

[Download WSDL](#)

This endpoint address can be compared against the endpoints in this [Microsoft document](#). They each map to a major Azure region. While Dynamics does not provide a means of determining your sub-region (such as US-East or US-West), this will allow you to at least know which major region your Dynamics instance is located in.

Since you cannot change or configure your Dynamics region in any way, it is recommended instead that you move your Azure resources to a sub-region that falls within the same major region that your Dynamics instance is in.

Decreasing message size

Large message sizes when communicating over the Azure Relay to the execution service can result in lower performance and potentially failed requests if the size is too large. When testing performance, it is recommended to turn off 'Entity Loading Log Enabled', 'Rule Execution Log Enabled' and 'Return Full Entity Image' in the 'Rule Execution Service' section of the InRule config entity, as these settings can contribute significantly to the size of the message returned from the execution service.

Batch Processing

InRule rule execution is primarily oriented at the level of an individual transaction for a given entity context. This is relatively straight-forward when rules are based on an entity event or on-demand process -- for example, Validate Account or Qualify Lead. In other scenarios, there may be a desire to execute rules across multiple entities that are related to a logical parent at a point in time - for example a tranche of Loans or Leads from a Tradeshow. In this case, InRule's collection handling is well suited to load and execute rules against multiple entities based on the relationship to the parent rule entity. The multiple entity scenario works to a point. The primary consideration is that all of the entity data is processed in a single rules execution request that can result in the loading and saving of hundreds to thousands of entities.

If the entity count for a rule execution request is expected to be upwards of 1,000 to 10,000+ entities, it's advisable to reassess, measure throughput and potentially consider batch approaches. Unfortunately, neither Dynamics nor InRule have a one-size-fits-all solution for managing batch operations. However, in these scenarios, a single rule execution request can usually be broken into multiple requests by employing common batch solution techniques and tools. The end goal is to establish a solution where a batch process will manage the iteration across the primary entities (e.g. Leads for a Tradeshow) and issue the rule execution requests for either individual Leads or smaller groups of Leads.

Enable Background Compilation

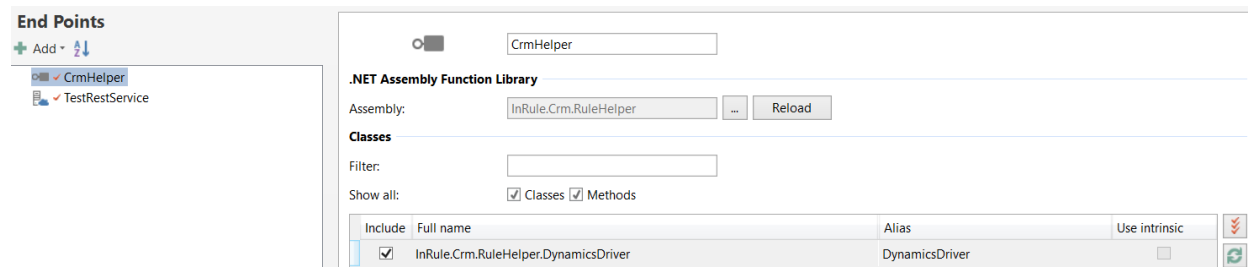
This setting controls whether the download, compilation, and caching of updates to rule applications from the catalog is handled on a background task. When enabled, rule execution will not be blocked while updating the rule app and will instead run immediately against the currently cached version. Self-hosted tenants should ensure this value is set to true for their deployment to avoid unnecessary catalog overhead. For additional information, please view the documentation available on the [Configuration Overview](#) page.

Disabling State Refresh

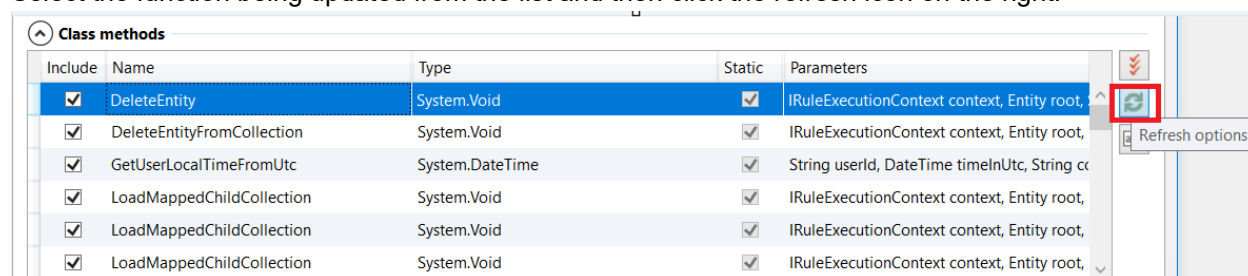
Disabling the state refresh option could improve performance for applications with many RuleHelper calls. State refresh is configured for both .Net Assembly Function Libraries, and User-Defined Functions.

.Net Function Assembly Libraries

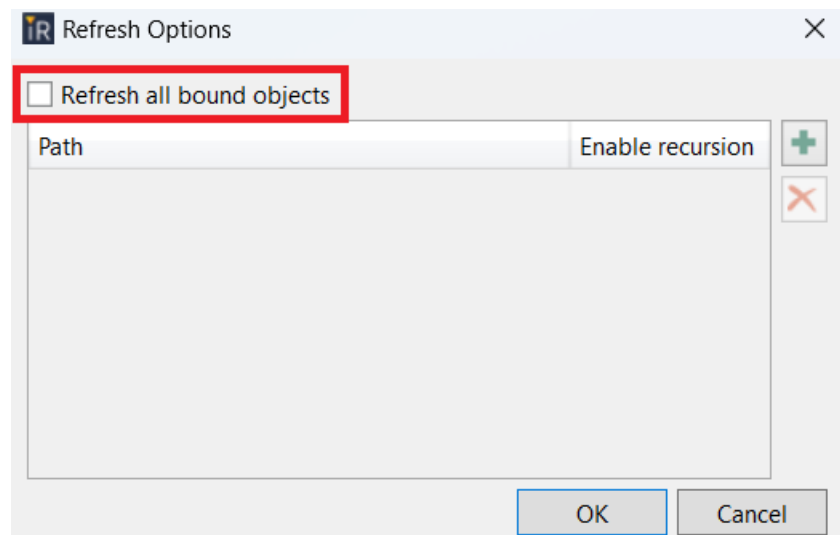
To disable state refresh for .Net Assembly Function Libraries, begin by navigating to the Endpoints tab in irAuthor. Once there, select the function library containing the function you wish to disable state refresh for.



Select the function being updated from the list and then click the refresh icon on the right.



A Refresh Options pop-up should appear – locate the “Refresh all bound objects” checkbox and uncheck it. Click OK.



State refresh should now be disabled for this function.

User-Defined Functions

To disable state refresh for User-Defined Functions, begin by navigating to the User-Defined Functions tab of your app in irAuthor. Once there, select the function you wish to disable state refresh for and click the “Modify state refresh options” link under the State Refresh Options section.

User Defined Function

Return type: None

Entity: None

Parameters:

Name	Data type	Description
FieldName	Text	

State refresh options

Manage options for refreshing state affected by this function.
[Modify state refresh options](#)

A State Refresh Options pop-up should appear. Deactivate the “Enable full refresh of all bound objects” checkbox and lick Ok.

State Refresh Options

☐ Enable full refresh of all bound objects

Path	Enable Recursion

OK Cancel

State refresh should now be disabled for this function.

Request and Response Limitations

Request Message Size

If the request message size is too large the execution service will return an `Internal Server Error`. You can enable WPF Tracing on the Rule Execution App Service to validate this is the issue.

Response Message Size

If the response message is too large the plugin will show this error message:

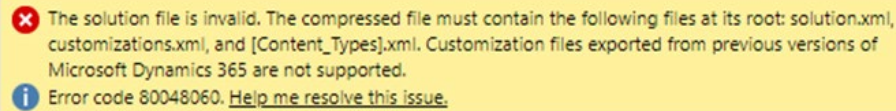
The maximum message size quota for incoming messages (xxxxxxx) has been exceeded. To increase the quota, use the MaxReceivedMessageSize property on the appropriate binding element. Please see Known Issues and Limitations in the inRule Deployment Guide

The response size limit can not be changed, since it is not configurable in Dynamics 365, but there are a few things you can do to reduce its size. In the relevant Rule Configuration settings, under the Rule

Execution Service section, you can disable the return of the entity image, rule execution log and entity loading log.

Deployment and Setup

Solution File Invalid Error



If you are encountering the error above, it is likely because you are attempting to import the InRule solution file using Dynamics' included Import Solution tool. This is not the recommended approach for upgrading the solution components. Because there are many operations involved in an upgrade beyond the Dynamics solution – preserving step registrations and SAS keys, applying security role permission sets, etc., all InRule solution installations **and upgrades** should be conducted using the approach outlined in [Section 3.3.3: InRule Solution for Dynamics 365](#).

Application Insights Location Error

Application Insights resources are not available in every region, the list of supported regions can be found in [Microsoft's Product Availability](#). By default the ARM template will attempt to deploy the App Insights resource in the resource group specified for the template deployment. If this resource group is in one of the unsupported regions you will get the following error:

```
New-AzureRmResourceGroupDeployment : 5:26:53 PM - Resource Microsoft.Insights/components
'ConnorArmTestAppInsights' failed with message '{
  "error": {
    "code": "MissingRegistrationForLocation",
    "message": "The subscription is not registered for the resource type 'components' in the location
'northcentralus'. Please re-register for this provider in order to have access to this location."
  }
}'
```

To fix this error we will have to choose a specific region for the Application Insights resource in the ARM template parameters file.

1. Locate InRule.Dynamics.Service.parameters.json

The ARM template parameters file is located in the *RuleExecutionAzureService* folder as, defined in [Section 3.3.2: Rule Execution App Service for Dynamics 365](#)

InRule for Microsoft Dynamics CRM Integration Framework > RuleExecutionAzureService				
	Name	Date modified	Type	Size
	azuredeploy.json	9/26/2018 12:49 PM	JSON File	6 KB
	azuredeploy.parameters.json	9/26/2018 12:49 PM	JSON File	1 KB
	InRule.Crm.WebJob.zip	9/26/2018 12:49 PM	Compressed (zipp...	4,974 KB
	Register-AzureApp.ps1	9/26/2018 1:48 PM	PS1 File	0 KB

2. Create an “applnsightsLocation” parameter

Open the file in your text editor of choice. First, create the “**applnsightsLocation**” parameter at the bottom of the parameters file. Set the value equal to a region where Application Insights resources are offered.

```
"appInsightsResourceName": {  
  "value": ""  
},  
"appInsightsLocation": {  
  {  
    "value": "SouthCentralUS"  
  }  
}
```

3. Save InRule.Dynamics.Service.parameters.json and continue deployment

Save and close the file. You can now proceed with the deployment process outlined in [Section 3.3.2: Rule Execution App Service for Dynamics 365](#) as normal; your rule execution app service will now deploy to the App Service Plan you defined in the steps above

- Should you encounter the following error repeatedly being logged in your AppService's Application Log:
Unhandled Exception: System.ServiceModel.AddressAlreadyInUseException: This endpoint requires IsDynamic = False
You need to delete the Azure Relay that your Webjob is attempting to connect to and redeploy it using the ARM Template included in the InRule deployment package.

Package Deployment Timeout

When running the deployment script from [InRule Solution for Dynamics 365](#), the Dynamics Package Deployer tool may timeout when communicating with Dynamics. This will result in a 'The request channel timed out while waiting for a reply' error. Depending on where in the process this error happens, the Package Deployer tool may retry or may stop execution. Once it completes, be sure to validate the service endpoint and custom action settings in [Appendix F: Rules Configuration and Settings](#), as incomplete deployments can reset these to default values.

AppSource Deployment Errors

Errors resulting from AppSource deployments can be difficult to debug, as the UI often provides very little information. The first thing to check is if there is any info in 'Solutions History', which can be found in advanced settings under the 'Customization' section. This page lists each solution install attempts, the result, and any errors from deployment. Issues at this step can sometimes arise from dependency issues or conflicts with other components installed in the instance. If the deployment fails before the solution install is even attempted, there will not be any attempts listed, and the only error will be the generic one displayed on the app management page. If this happens, attempt to install the app at a later time, and ensure that all of the appropriate licenses are provisioned. If this still does not resolve the issue, contacting Microsoft support may be required.

Miscellaneous Troubleshooting Items

- Relay Provider Error** - When deploying the Azure WCF Relay (aka Azure Relay), you will need to ensure the Azure subscription you are deploying the Relay into has the Relay provider enabled. This can sometimes happen with older subscriptions that have not used relays before. For more information about this issue, refer to the link [here](#)

- **Plugin Isolation for On-Prem** - When deploying plugins without isolation in an on-prem environment, Dynamics requires that the user registering the plugin must be added as a Deployment Administrator from Deployment Manager. If the registering user lacks the proper permissions, when deploying the package Dynamics will return an error stating **“Assembly must be registered in isolation.”**

